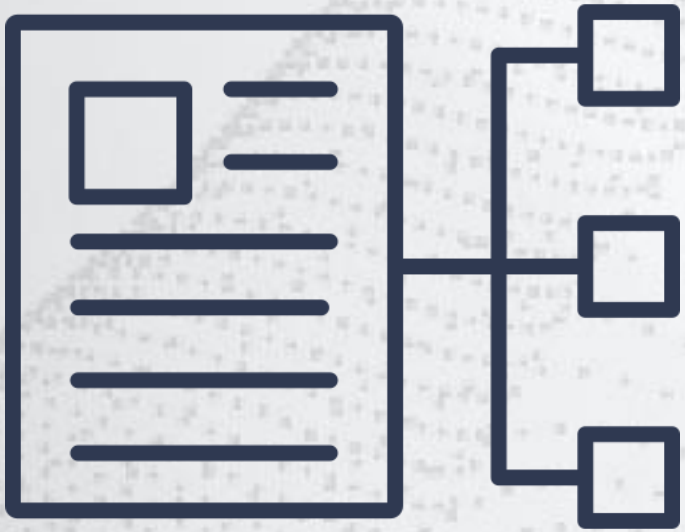
KIBERNUM
IT Academy

Módulo 7 – Fundamentos de Deep Learning

Introducción a las redes neuronales

Especialización en Ciencia de Datos



Objetivos

Describir los conceptos fundamentales las redes neuronales y su utilidad para la resolución de problemas de aprendizaje de máquina.

Contenido

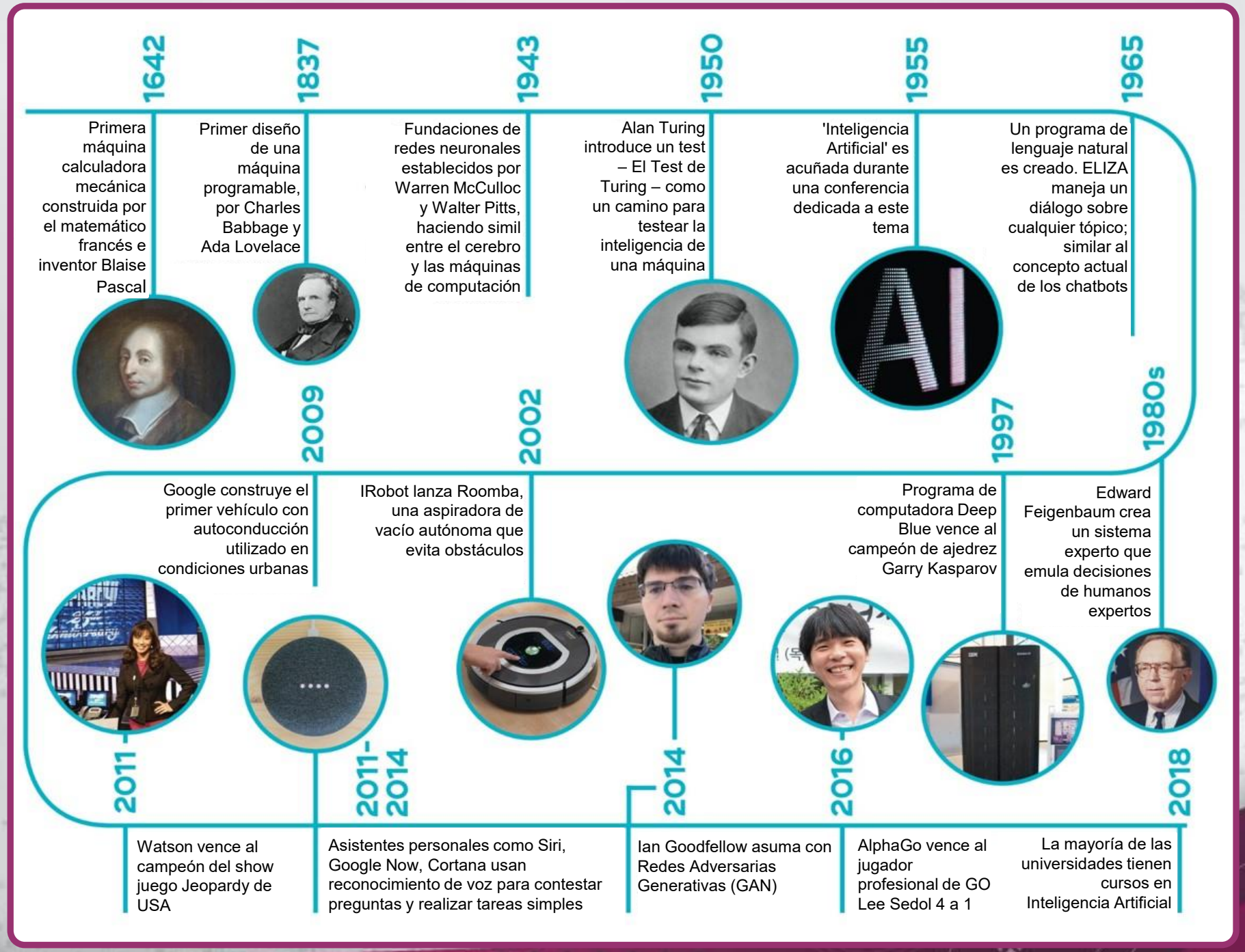
1. Un poco de Historia.
2. Qué es una red neuronal.
3. El Perceptrón.
4. Implementando un perceptrón.

Un Poco de Historia

Un Poco de Historia

Historia de la IA: Frank Rosenblatt y el Mark I Perceptrón, el primer ordenador fabricado específicamente para crear redes neuronales en 1957.

Cronología y Evolución

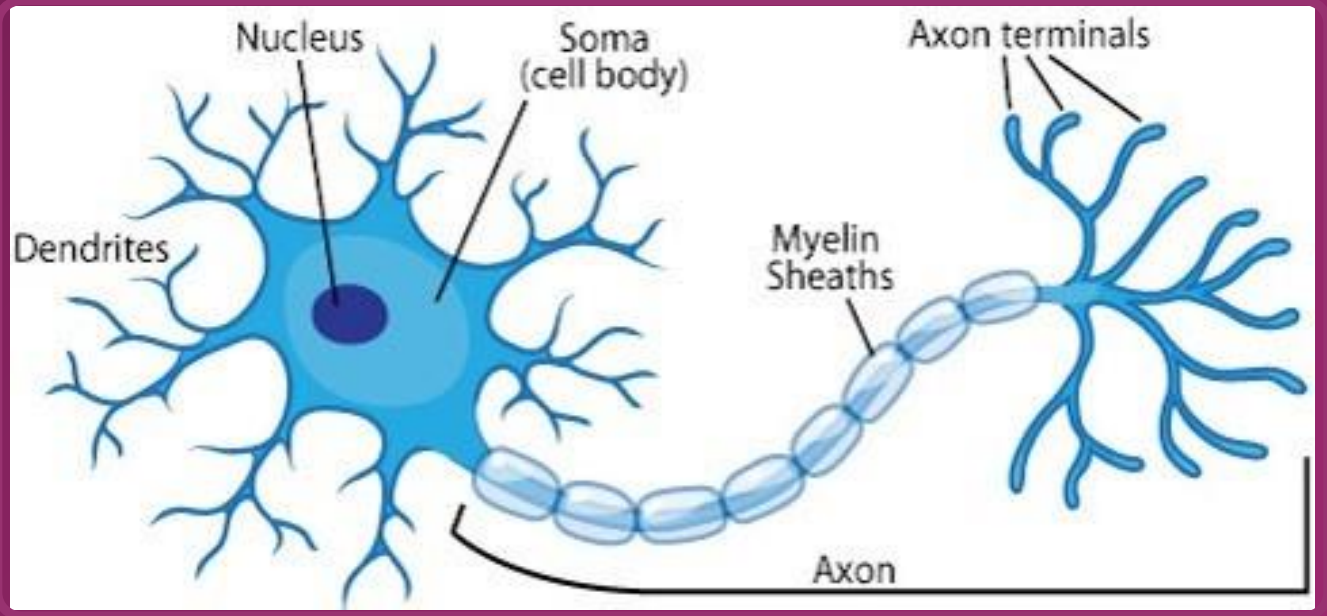


KIBERNUM

¿Qué es una Red Neuronal?

¿Qué es una Red Neuronal?

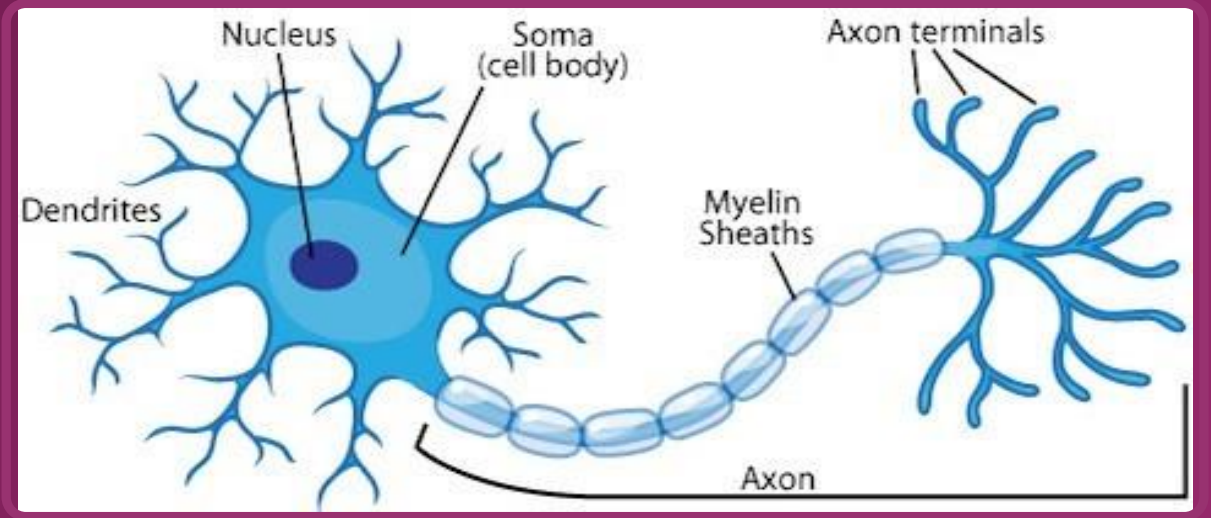
Es un paradigma de programación hermosamente inspirado en la biología que permite a las máquinas aprender a partir de datos observados.
Redes Neuronales y Deep Learning ofrecen las mejores soluciones para el reconocimiento de imágenes, procesamiento del lenguaje natural, reconocimiento de voz, entre otros.

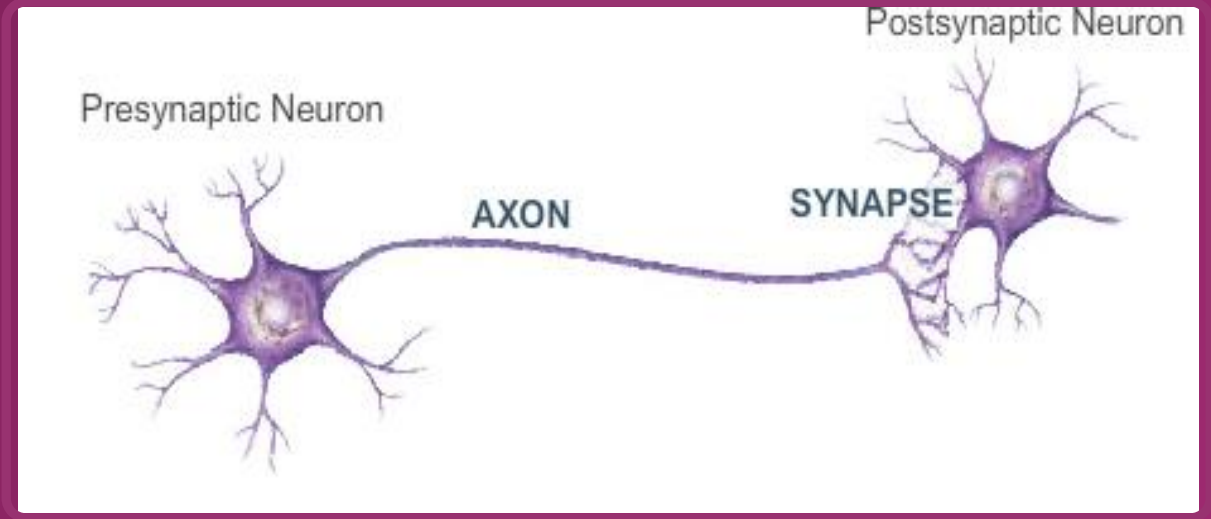


Conexiones Neuronales

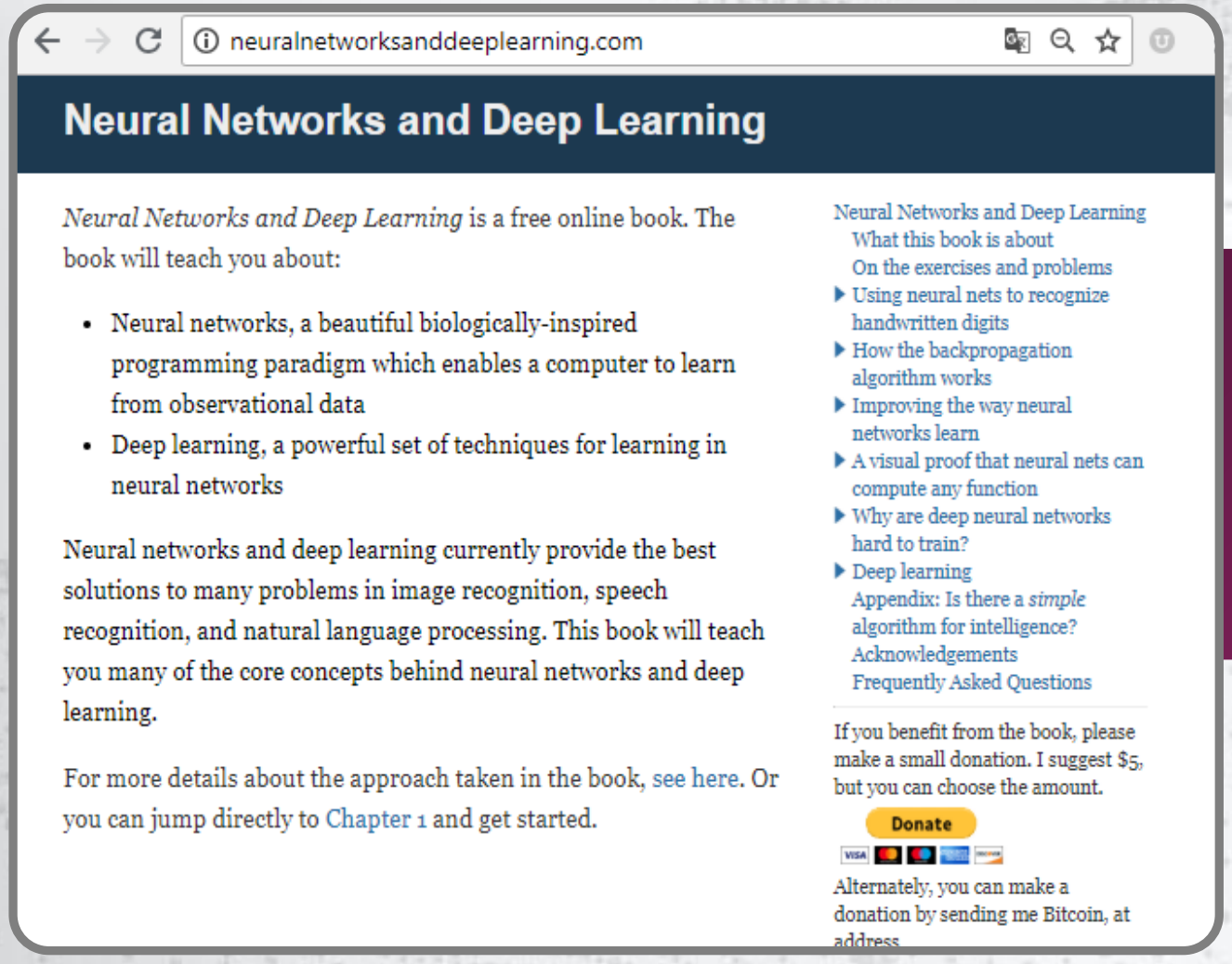
Las comunicaciones neuronales dependen de señales eléctricas y químicas. Una neurona genera una señal eléctrica que es conducida por el Axon.

Las Dendritas reciben las señales eléctricas, y basados en estas señales de entrada, las conducen a otras neuronas.





Bibliografía Recomendada



Recomendamos esta web en donde puedes bajar un libro gratis de Deep learning:

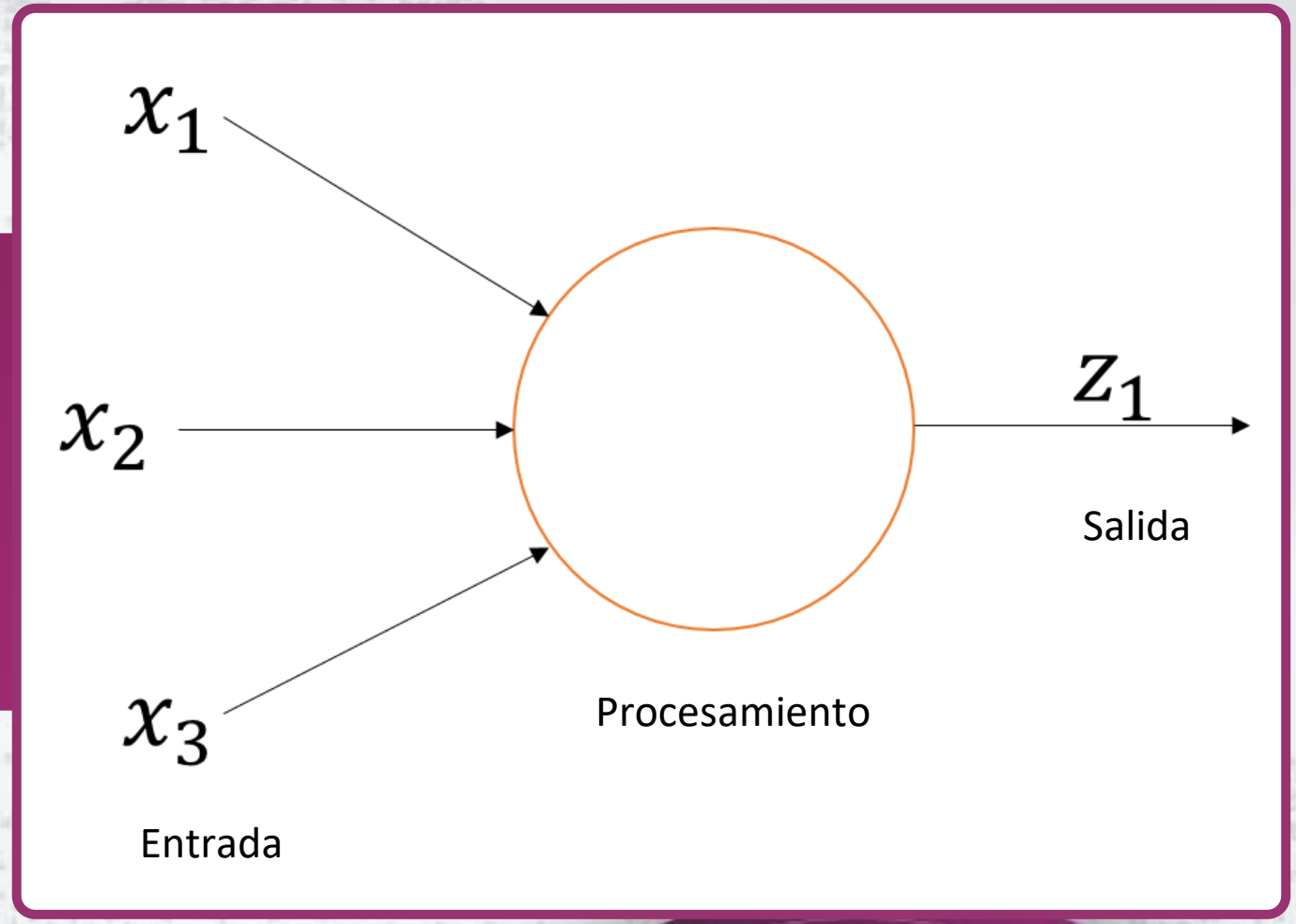
http://neuralnetworksanddeeplearning.com

KIBERNUM

El Perceptrón

El Perceptrón

El Perceptrón es una metáfora de una neurona, de tipo artificial, desarrollado en 1950 por el científico Frank Rosenblatt.
Un perceptrón toma varias entradas (x1, …, xn) y devuelve una sola salida, mediante un procesamiento.



Ejemplo de un Perceptrón

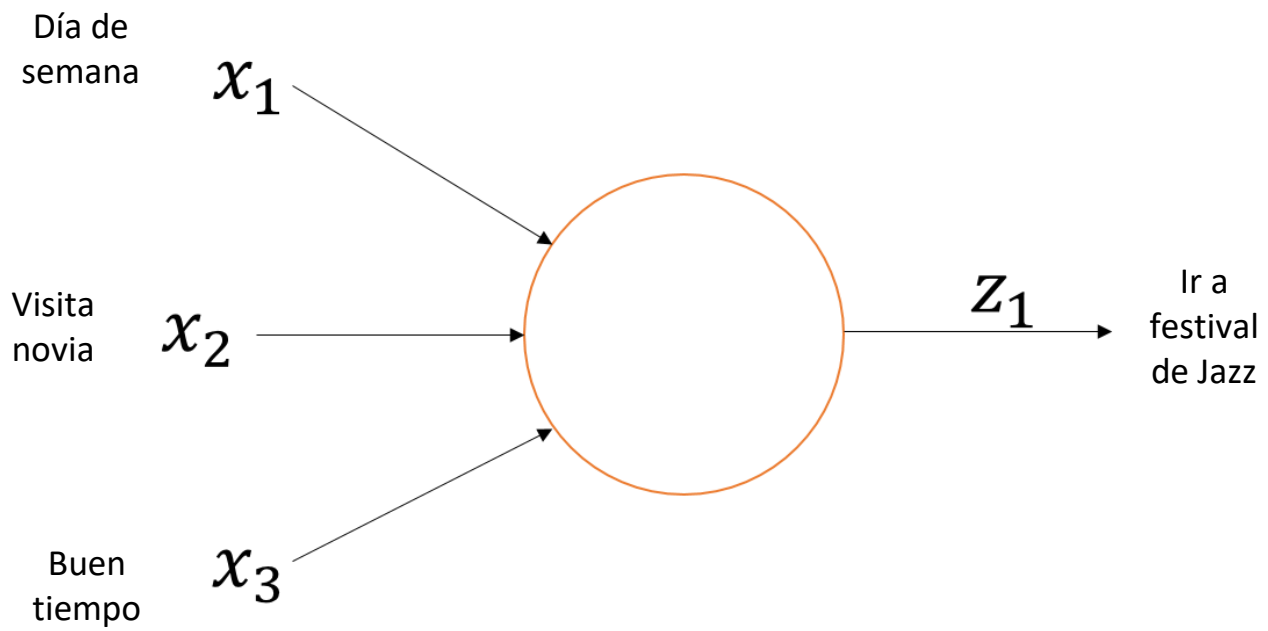
Modelaremos un perceptrón para representar la siguiente situación. Suponga que:

x1: es la variable que indica si es día de semana o fin de semana (0 día de semana, 1 fin de semana).

x2: es la variable que indica si tu novia te visitará (0 no te visitará, 1 sí te visitará).

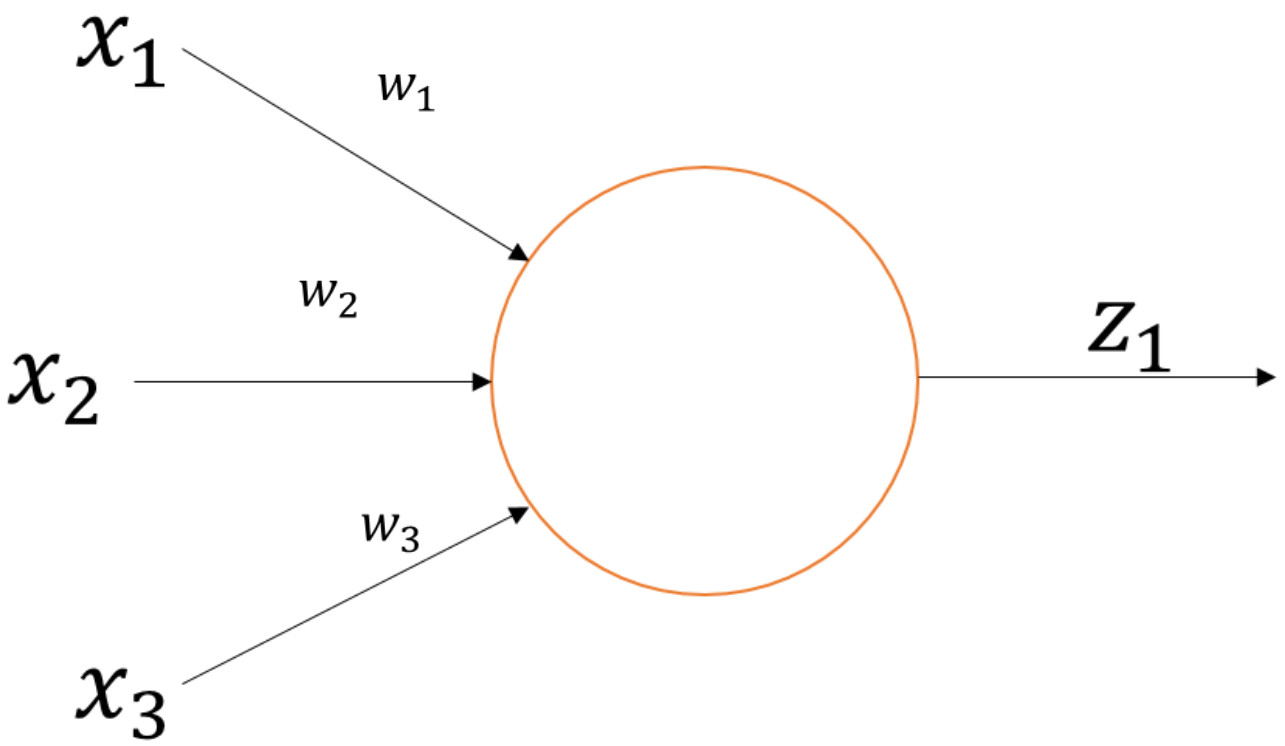
x3: es la variable que indica si hay buen tiempo o mal tiempo (0 mal tiempo, 1 buen tiempo).

Output: es la respuesta acerca si irás al festival de Jazz de la ciudad. (0 no vas, 1 si vas)



Asignando Pesos

Cada factor o variable tendrá un mayor o menor peso en la decisión que tomarás. Para esto, introduciremos en el modelo los valores de w_1, w_2 y w_3 que son números reales y expresan la importancia de cada variable en la decisión.

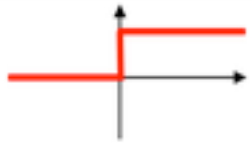
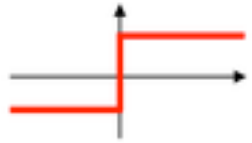
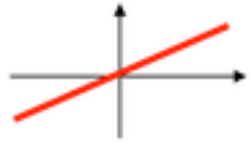
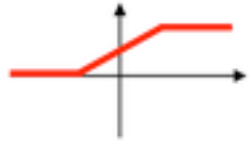
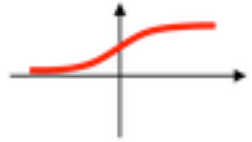
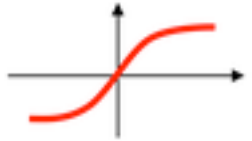


Decisión

La decisión tomada en la salida corresponderá a si la suma de los pesos por valores de las variables es mayor o menor que un umbral. El umbral también es un parámetro numérico.

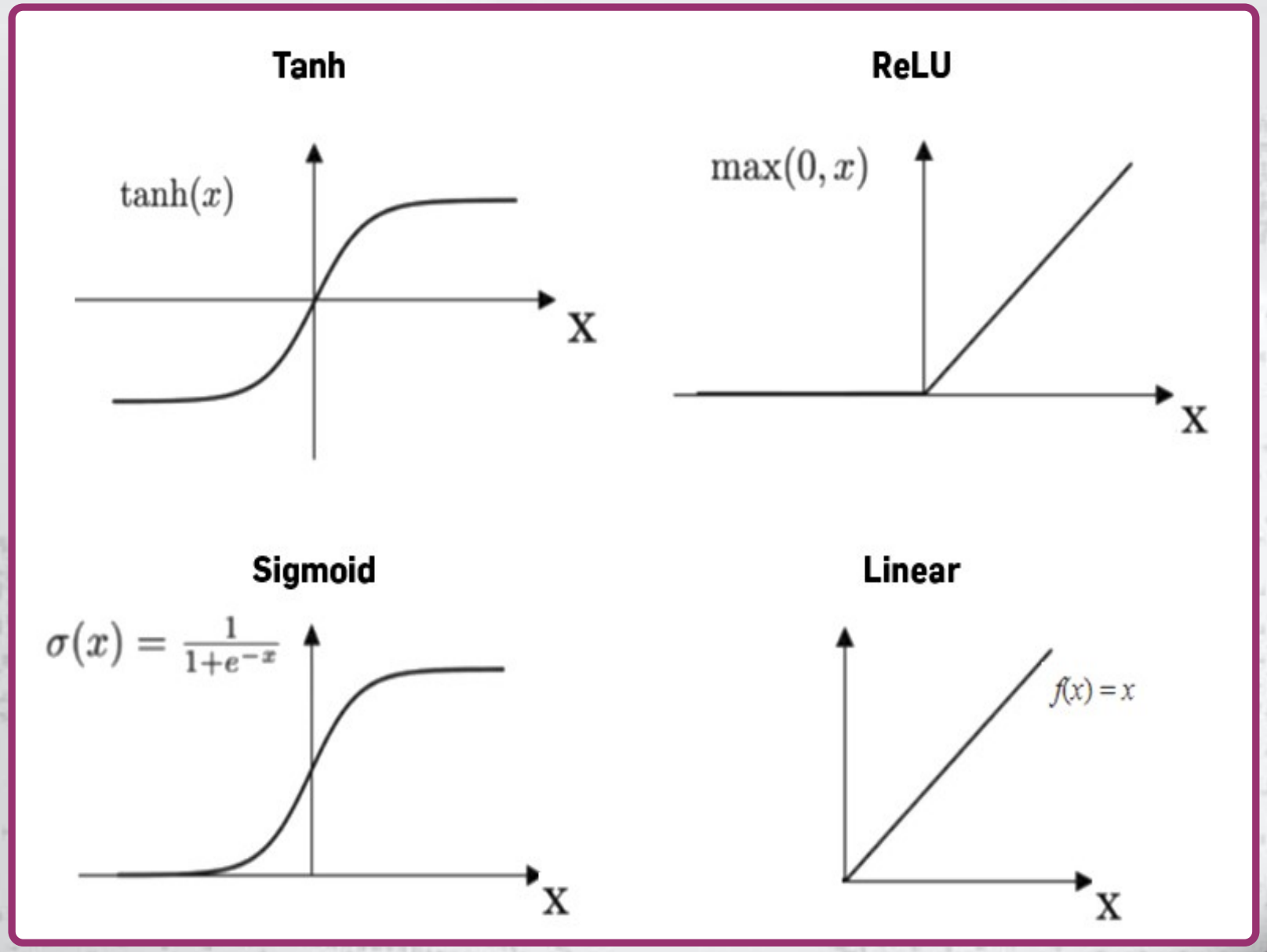
$$salida = \begin{cases} 0 & , \quad \sum w_j x_j \leq umbral \\ 1 & , \quad \sum w_j x_j > umbral \end{cases}$$

Función de Activación

Función de Activación	Ecuación	Ejemplo	Gráfico 1-D
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer NN	

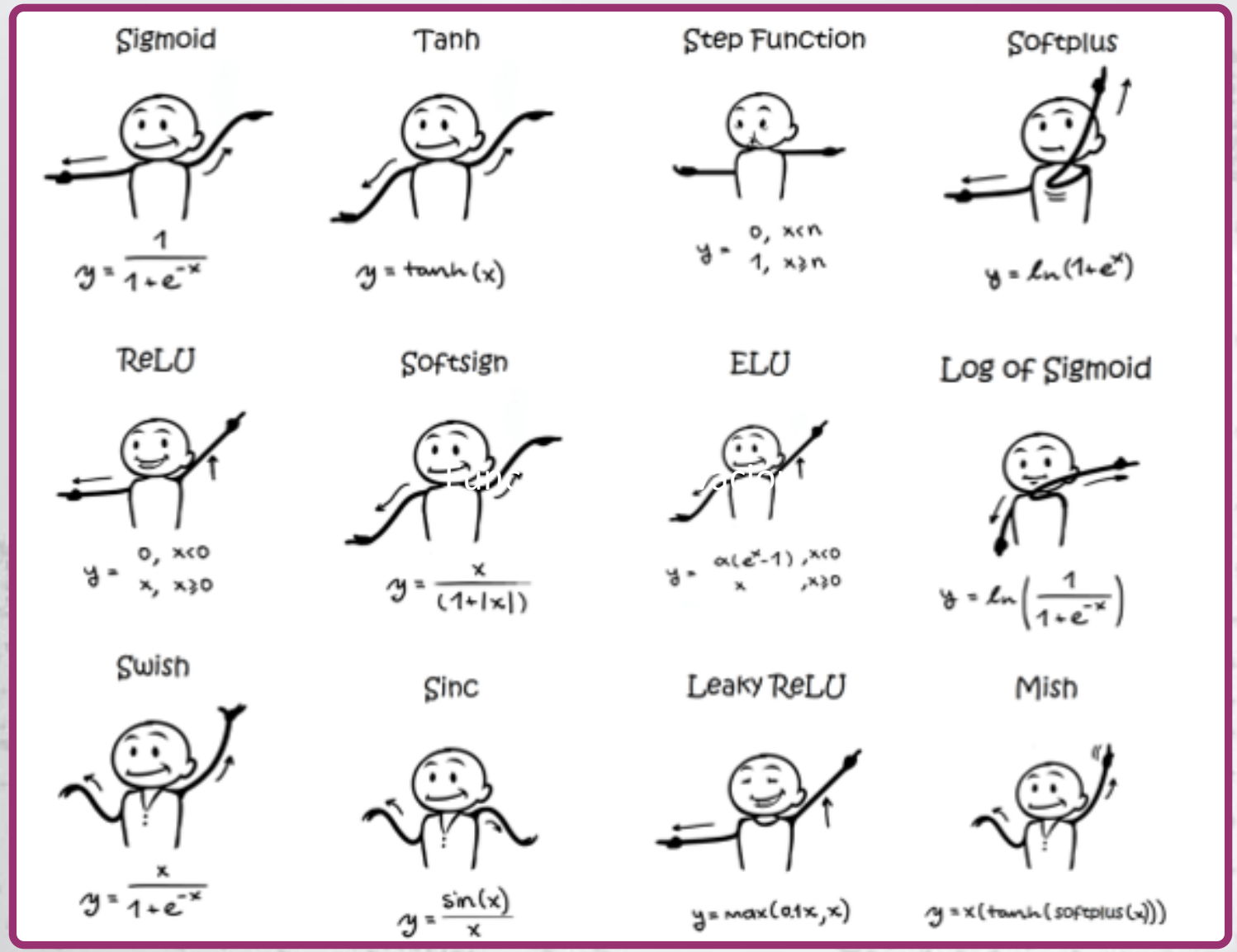
La función mostrada anteriormente es la más simple, pero en realidad hay varias funciones de activación que se podrían utilizar para generar la salida a partir de las variables y los pesos asignados.

Función de Activación



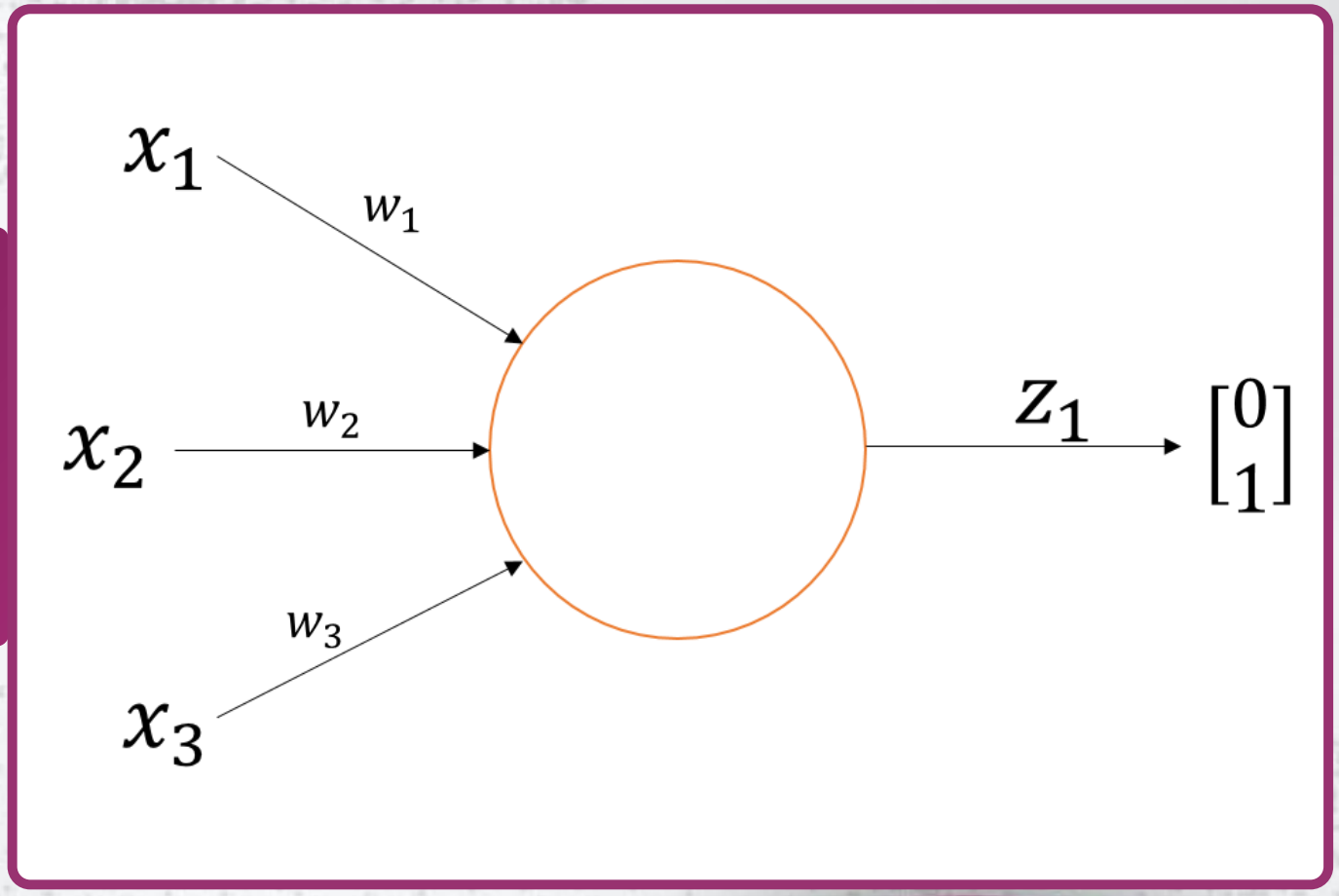
Estas son las funciones de activación más comunes:

Función de Activación



Función de Activación

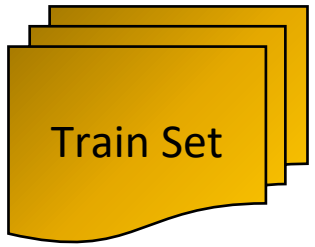
Si buscamos que una neurona retorne un valor binario (es decir 0 ó 1), ¿qué tipo de función de activación podemos utilizar?

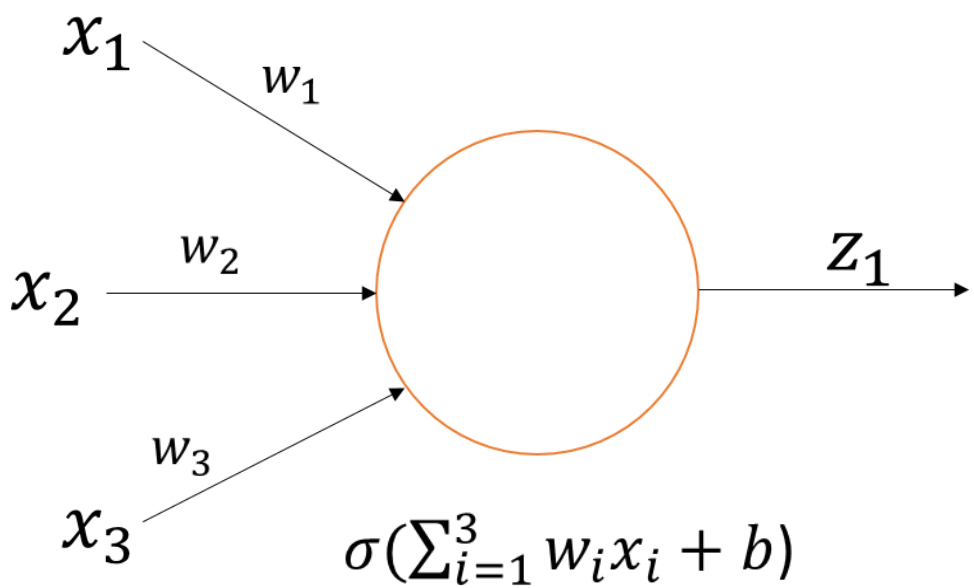


El Perceptrón

El Perceptrón trabaja de la siguiente forma:

x1	x2	x3
1.5	4.8	2.2
3.4	0.6	1.7
-2.3	7.2	-0.2

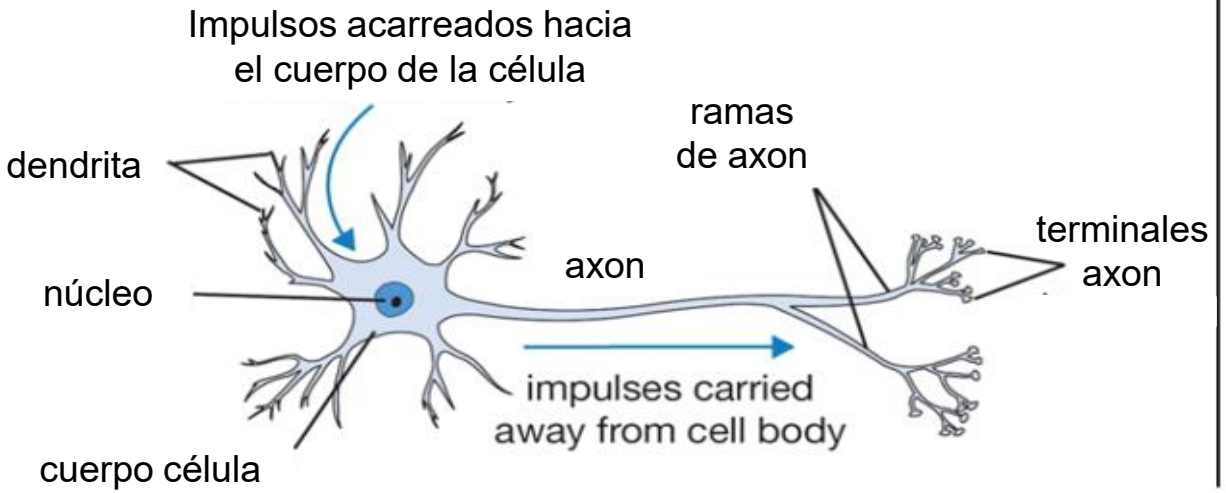




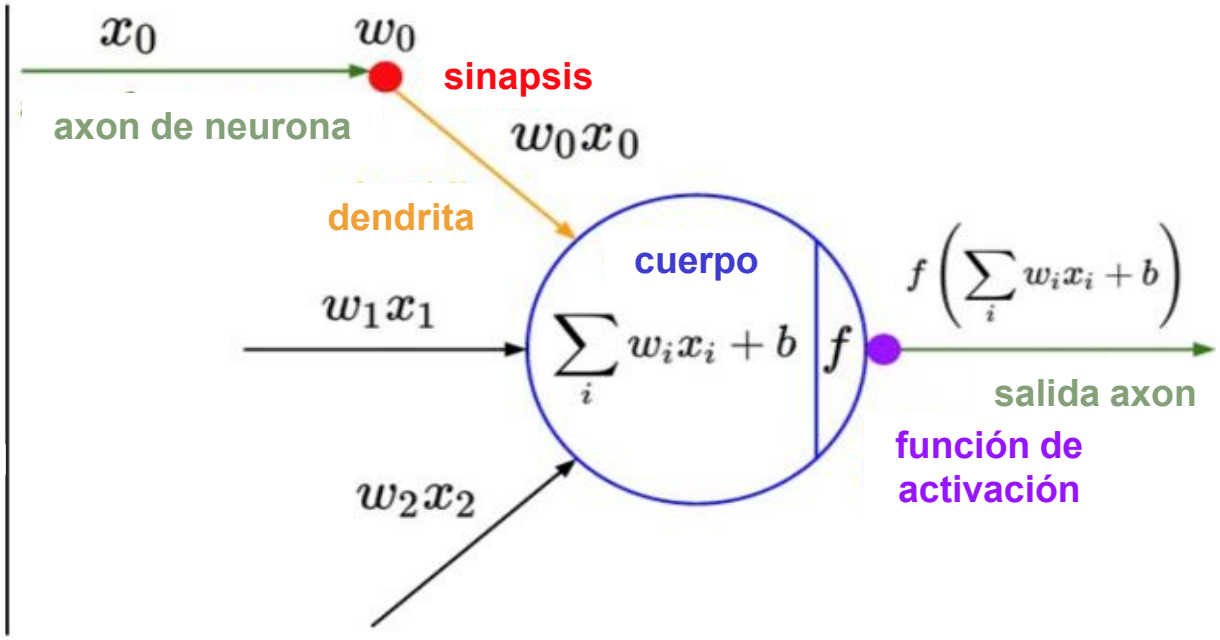
1. Recibe un input → 2. Se asigna peso a cada input → 3. Computa los inputs → 4. Genera la respuesta

Metáfora

Acá podemos observar la analogía entre una neurona y un perceptrón.



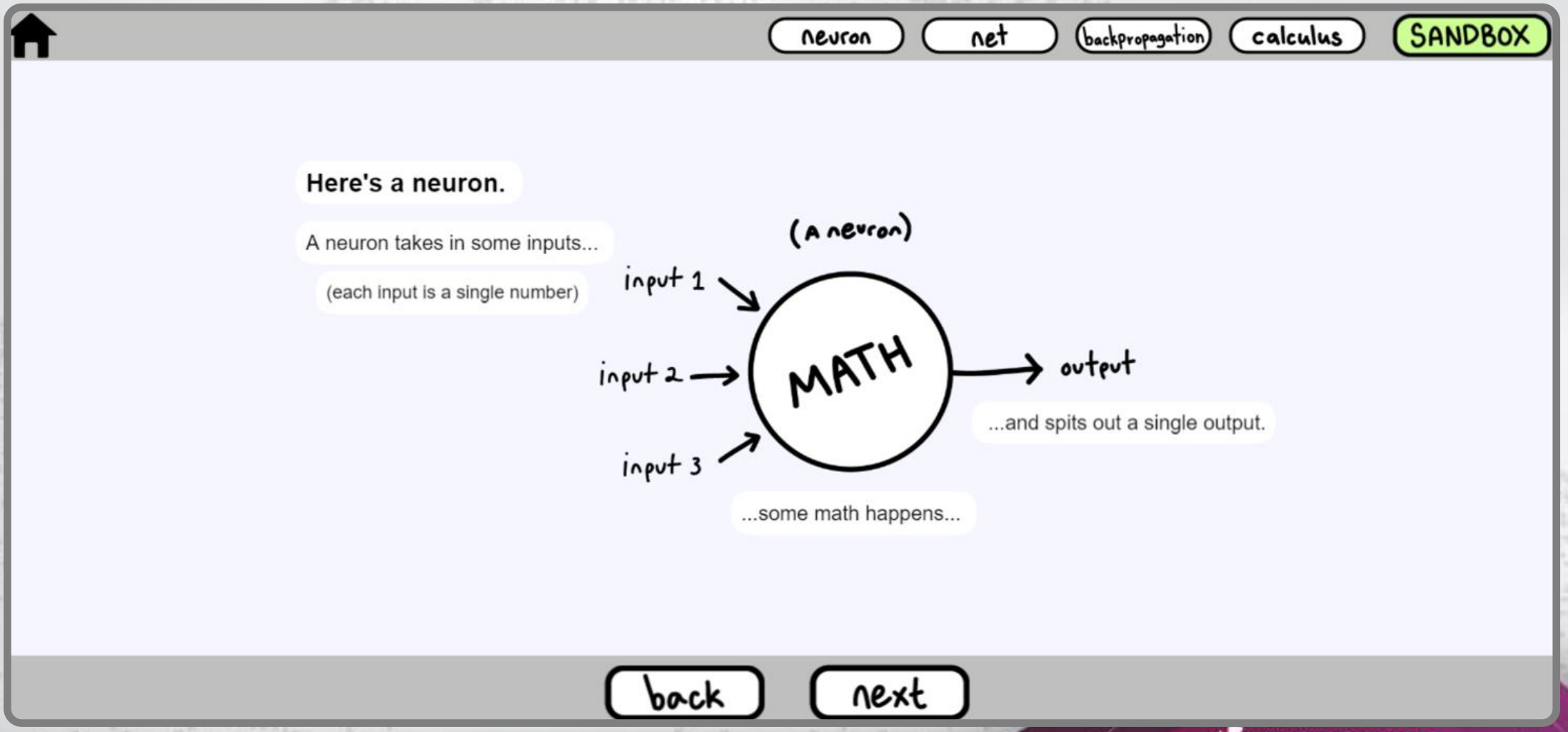
Representación biológica de una neurona



Representación matemática de una neurona

Playground

https://aegeorge42.github.io/



Implementación de un Perceptrón

Implementando un Perceptrón

Imagina que contamos con datos del comportamiento de una persona respecto a sus decisiones de ir a jugar tenis el sábado por la mañana.

Implementando un Perceptrón

Esta es la información reunida de su comportamiento de los últimos sábados. Ahora el desafío es construir un perceptrón que aprenda y logre emular las decisiones.

tiempo	temperatura	humedad	viento	Juega Tenis
Soleado	Calor	Alta	Debil	No
Soleado	Calor	Alta	Fuerte	No
Nublado	Calor	Alta	Debil	Si
Lluvia	Agradable	Alta	Debil	Si
Lluvia	Frio	Normal	Debil	Si
Lluvia	Frio	Normal	Fuerte	No
Nublado	Frio	Normal	Fuerte	Si
Soleado	Agradable	Alta	Debil	No
Soleado	Frio	Normal	Debil	Si
Lluvia	Agradable	Normal	Debil	Si
Soleado	Agradable	Normal	Fuerte	Si
Nublado	Agradable	Alta	Fuerte	Si
Nublado	Calor	Normal	Debil	Si
Lluvia	Agradable	Alta	Fuerte	No

Implementando un Perceptrón

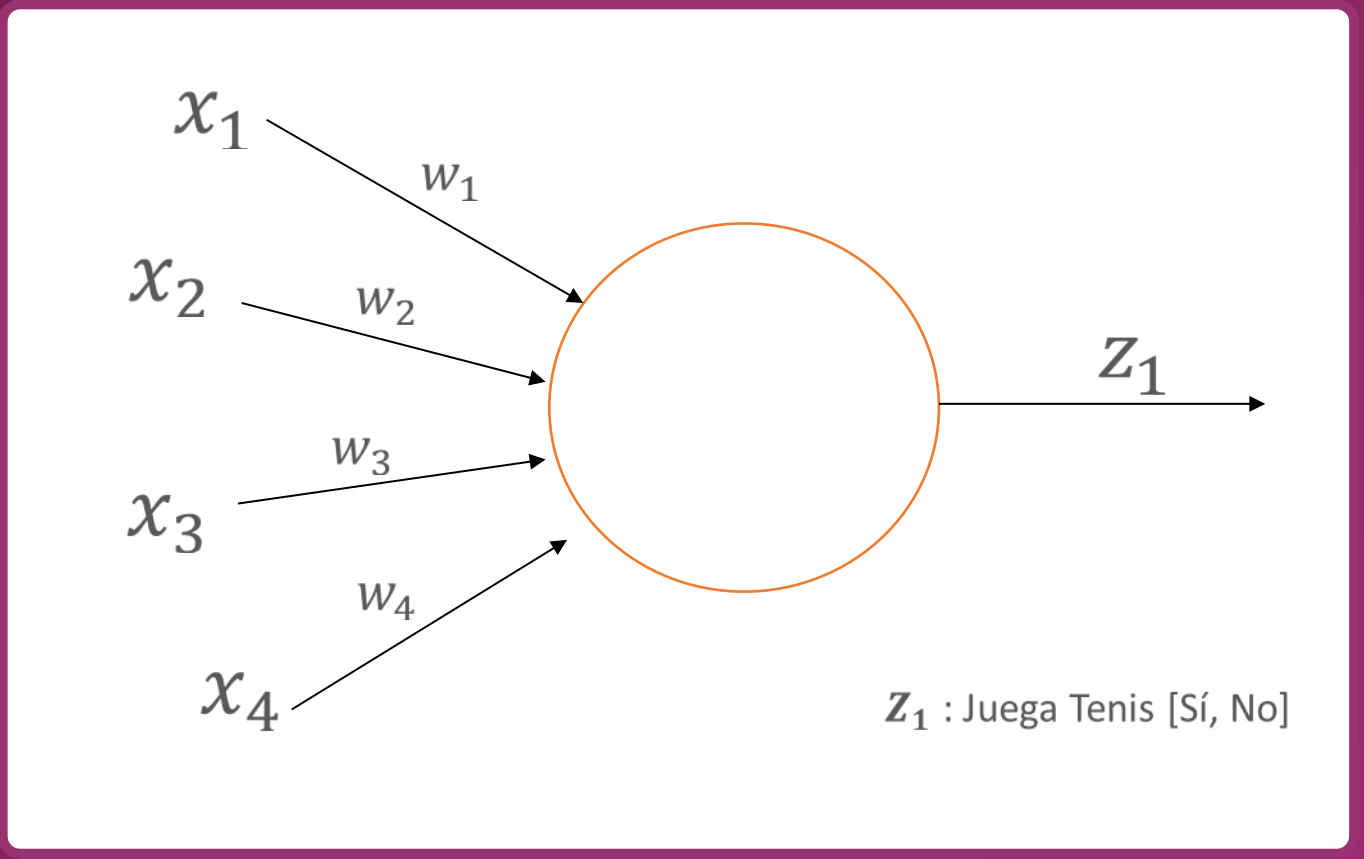
El siguiente es un perceptrón con 4 entradas y una salida. En donde:

x_1 : tiempo [soleado, nublado, lluvia]

x_2 : temperatura [calor, agradable, frío]

x_3 : humedad [alta, normal]

x_4 : viento [débil, fuerte]

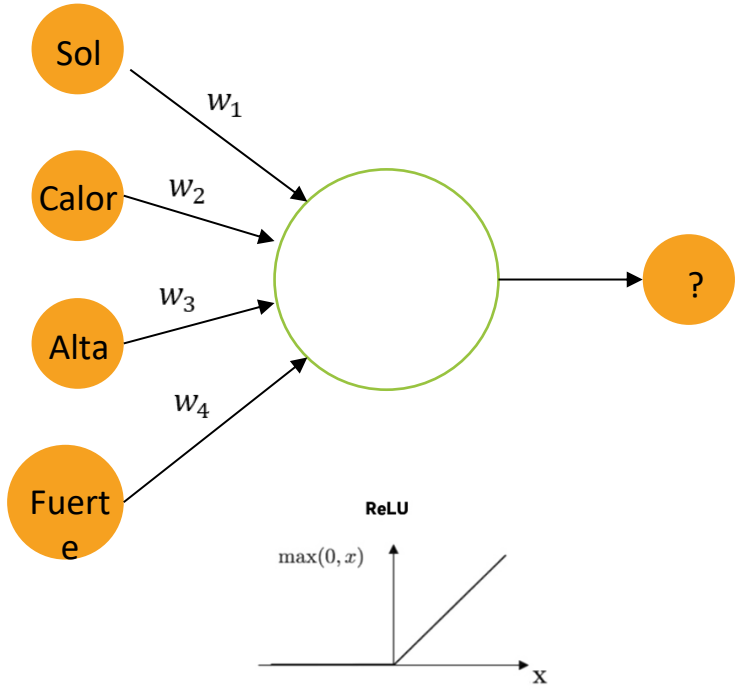


Implementando un Perceptrón

Entonces, haremos lo siguiente:

1. Fijaremos valores arbitrarios de w.
2. Ingresaremos cada uno de los datos N registros al perceptrón y calcularemos la salida en cada caso (epoch).
3. Compararemos las salidas del perceptrón con el valor real de cada registro para determinar una métrica de error.
4. Ajustaremos los valores de w y volvemos al punto 2 para repetir el proceso.
5. Cuando hayamos completado una cantidad definida de epochs nos detenemos.

tiempo	temperatura	humedad	viento	Juega Tenis
Soleado	Calor	Alta	Debil	No
Soleado	Calor	Alta	Fuerte	No
Nublado	Calor	Alta	Debil	Si
Lluvia	Agradable	Alta	Debil	Si
Lluvia	Frio	Normal	Debil	Si
Lluvia	Frio	Normal	Fuerte	No
Nublado	Frio	Normal	Fuerte	Si
Soleado	Agradable	Alta	Debil	No
Soleado	Frio	Normal	Debil	Si
Lluvia	Agradable	Normal	Debil	Si
Soleado	Agradable	Normal	Fuerte	Si
Nublado	Agradable	Alta	Fuerte	Si
Nublado	Calor	Normal	Debil	Si
Lluvia	Agradable	Alta	Fuerte	No



Juega Tenis
No
No
Si
Si
Si
No
Si
No
Si
Si
Si
Si
Si
No

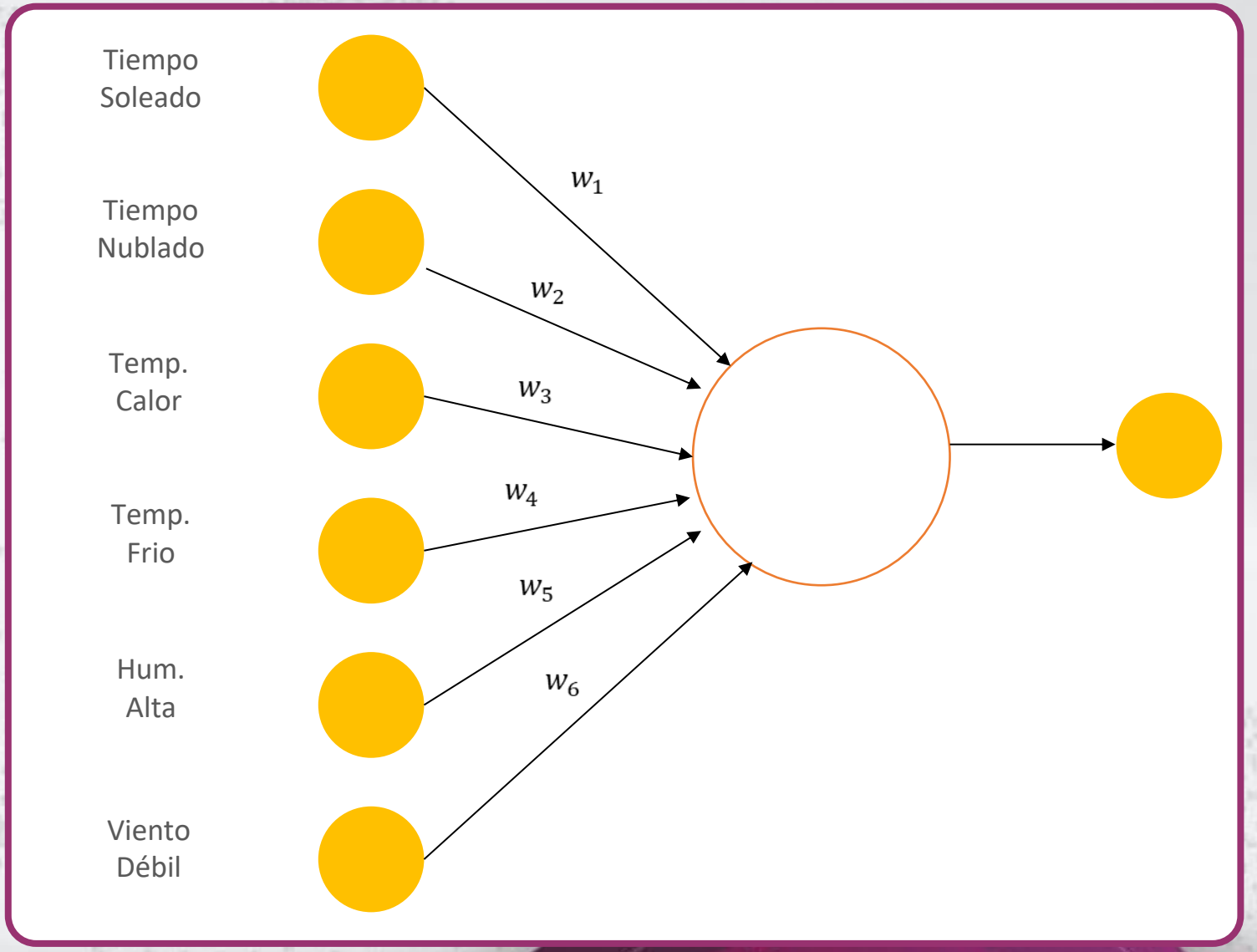
Implementando un Perceptrón

Sin embargo, debemos codificar las variables categóricas. Así que haremos la siguiente modificación:

Tiempo Soleado	Tiempo Nublado	Temperatura Calor	Temperatura Frio	Humedad Alta	Viento Debil	Juega Tenis
1	0	1	0	1	1	0
1	0	1	0	1	0	0
0	1	1	0	1	1	1
0	0	0	0	1	1	1
0	0	0	1	0	1	1
0	0	0	1	0	0	0
0	1	0	1	0	0	1
1	0	0	0	1	1	0
1	0	0	1	0	1	1
0	0	0	0	0	1	1
1	0	0	0	0	0	1
0	1	0	0	1	0	1
0	1	1	0	0	1	1
0	0	0	0	1	0	0

Implementando un Perceptrón

Entonces, nuestro perceptrón tendría 6 nodos de entrada y 1 salida.



Implementando un Perceptrón

Definimos algunas funciones convenientes que utilizaremos más adelante:

```python
# Define una función que calcule el producto punto entre dos vectores:
def dot_product(x, w):
    return np.dot(x, w)
```

Python

```python
# Define la función de activación.
def activation(x):
    # En este caso, utilizaremos la función escalón (step function):
    return 1 if x >= 0 else 0
```

Python

Implementando un Perceptrón

```python
# Define la función de entrenamiento del perceptrón:
def perceptron_train(X, y, num_epochs, learning_rate):

    # Inicializa los pesos aleatoriamente
    weights = np.random.rand(X.shape[1])

    errors = []

    # iterar las epocas
    for epoch in range(num_epochs):

        epoch_error = 0

        # Itera sobre todos los ejemplos de entrenamiento
        for i in range(X.shape[0]):

            # Calcula la salida del perceptrón
            output = activation(dot_product(X[i], weights))

            # Actualiza los pesos si la salida es incorrecta
            if output != y[i]:
                error = y[i] - output
                weights += learning_rate * error * X[i]
                epoch_error += abs(error)

        errors.append(epoch_error)

    return weights, errors
```

Esta función es la que realizará el entrenamiento.

Implementando un Perceptrón

Ejecutamos el entrenamiento y obtenemos los pesos w.

```python
# Ahora puedes utilizar la función perceptron_train para entrenar el perceptrón con tus datos de entrenamiento:
# En este ejemplo, utilizamos los datos de entrenamiento X y y para entrenar el perceptrón durante 100 épocas,
# con una tasa de aprendizaje de 0.1. La función devuelve los pesos entrenados.
weights, errors = perceptron_train(X, y, num_epochs=30, learning_rate=0.1)
```

Python Python

```python
# estos son los pesos calculados
weights
```

Python Python

```
array([ 0.05290118,  0.94498236,  0.1164077 , -0.05911127, -0.44112394,
        0.26115013])
```

Implementando un Perceptrón

Ahora probamos los pesos del modelo para hacer una predicción sobre un ejemplo.

```python
# Para hacer predicciones con el perceptrón entrenado, utiliza la siguiente función:
def perceptron_predict(x, weights):
    return activation(dot_product(x, weights))
```

Python Python

```python
# Puedes utilizarla para predecir la salida de un ejemplo dado:
x = np.array([0, 1, 0, 0 ,1, 0])
prediction = perceptron_predict(x, weights)
print(prediction)
```

Python

```
1
```

Implementando un perceptrón

Evaluamos el desempeño:

```python
# hacemos predicciones sobre el set X, por simplicidad no haremos cross validation,
# pero debieramos siempre
y_pred = [perceptron_predict(x, weights) for x in X]
y_pred
```

Python

```
[0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0]
```

```python
sum(y_pred == y)/len(y)
```

Python

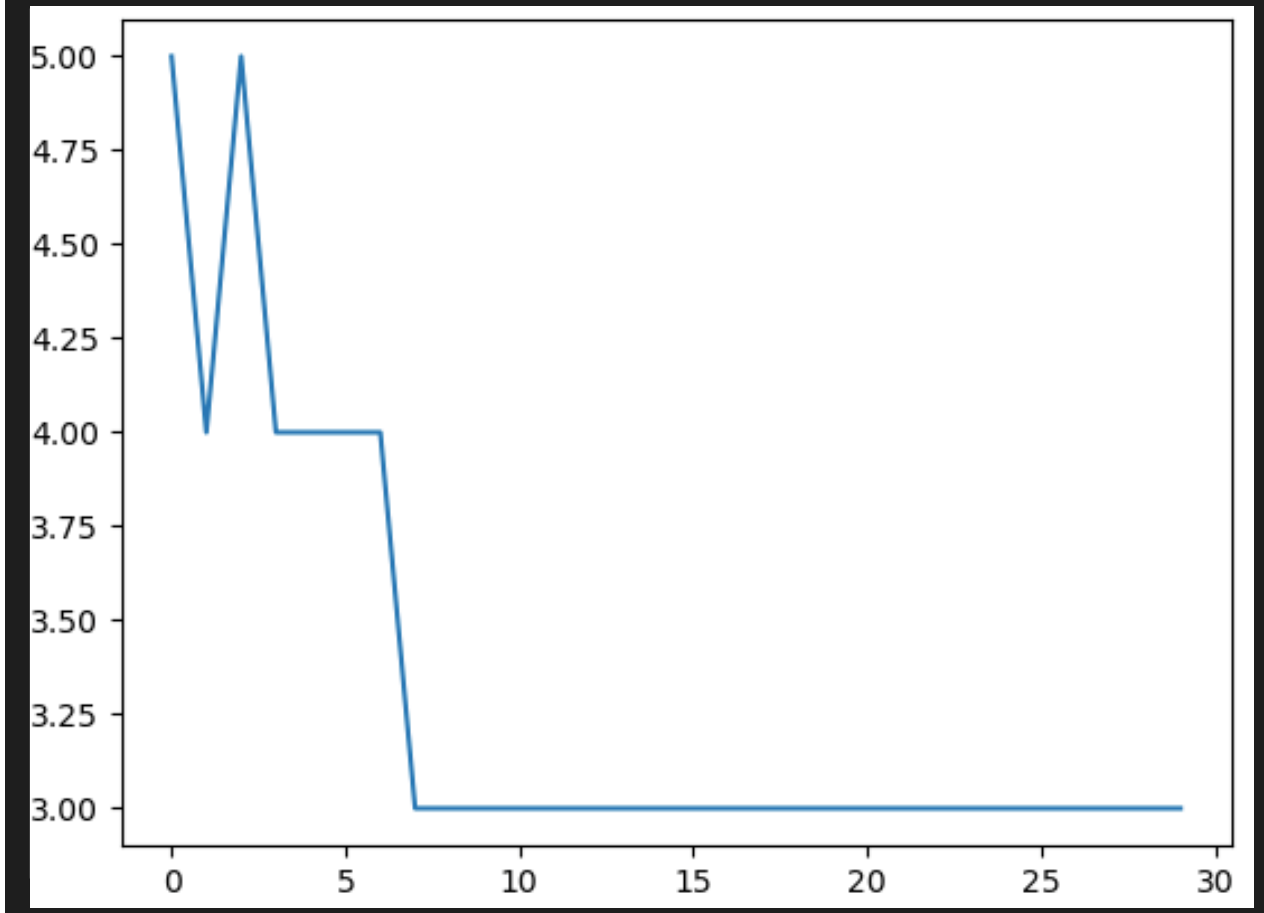
```
0.9285714285714286
```

Implementando un Perceptrón

```python
plt.plot(errors)
```

Python

```
[<matplotlib.lines.Line2D at 0x23b6d11b100>]
```



Graficamos el error por epoch. Nótese cómo después de algunos epochs el error disminuye y se estabiliza en un valor fijo.

Dudas y consultas

KIBERNUM

Fin Presentación

KIBERNUM