TALLER TEORICO PRACTICO EN ESTRUCTURAS DE DATOS Y APRENDIZAJE PROFUNDO

Fabian Perez - Semillero HDSP

Henry Mantilla - Semillero HDSP

High-Dimensional Signal Processing Research Group

¿Qué entendemos por la palabra datos?

"Se pueden tener datos sin información pero no se puede tener información sin datos"

VISIÓN POR COMPUTADOR

Classification

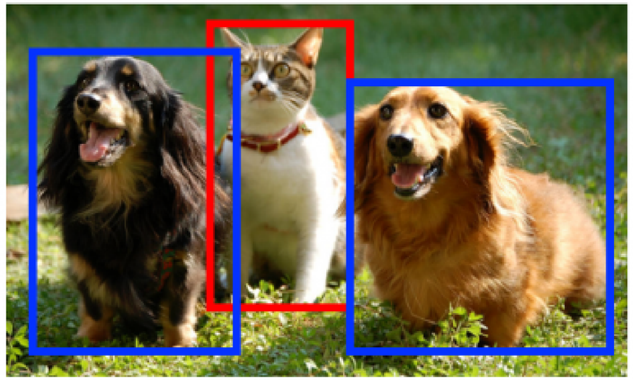
CAT

Classification + Localization

CAT

Single object

Object Detection



CAT, DOG

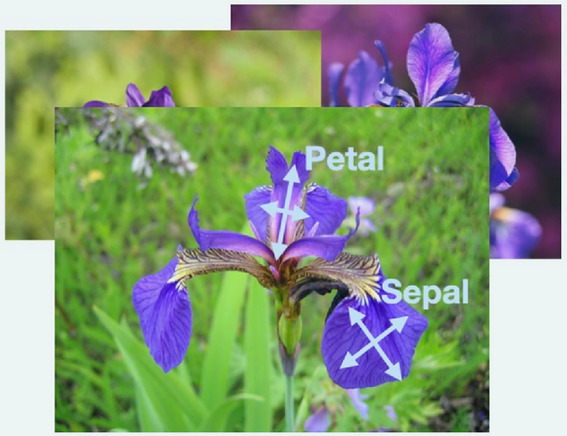
Instance Segmentation

CAT, DOG

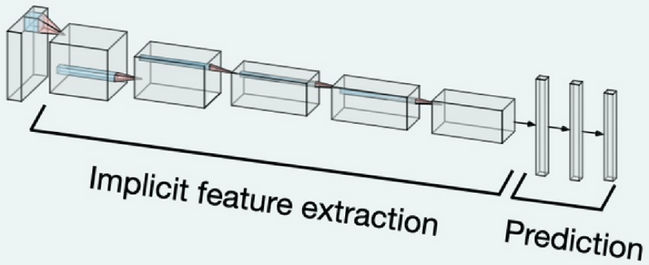
Multiple objects

Datos estructurados y no estructurados

Unstructured data



Deep learning

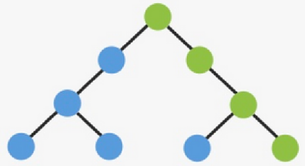


“Manual” feature extraction

Structured data

	Sepal length	Sepal width	Petal length	Petal width
Flower 1	5.1	3.5	1.4	0.3
Flower 2	4.9	3.0	1.4	0.2
Flower 3	5.9	3.0	5.1	1.8
...	...	...	...	...

Conventional machine learning

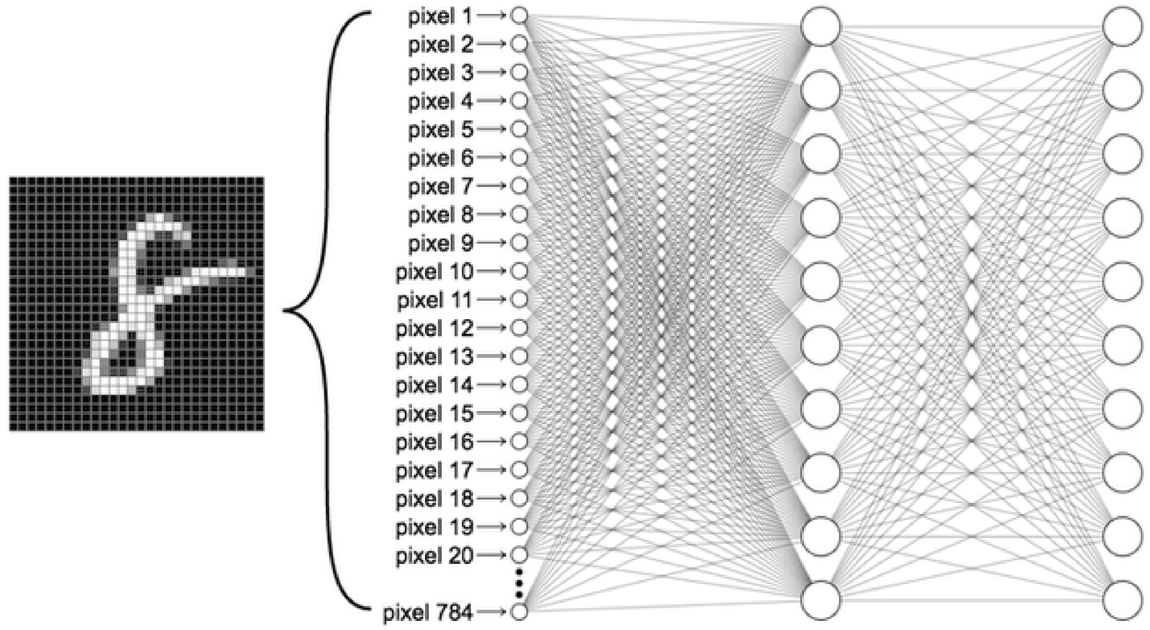


Estructuras de datos usadas en aprendizaje profundo

Vectores

Un vector es un arreglo de valores numéricos, cada componente del vector es identificada mediante su indice. El numero de componentes corresponde a la dimensionalidad del vector.

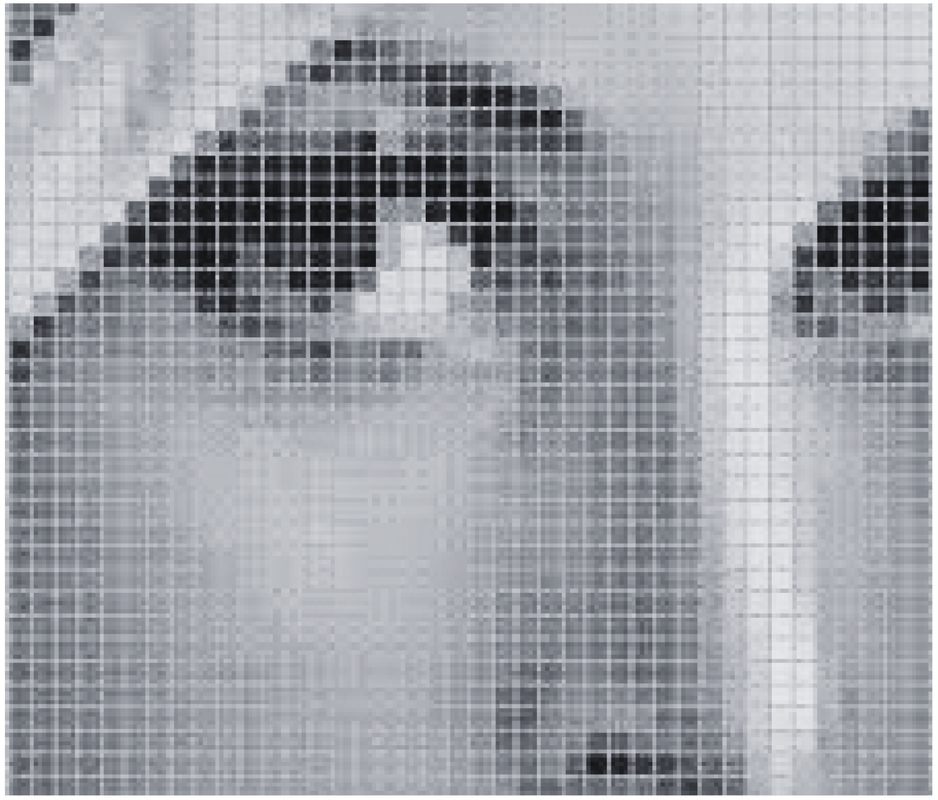
$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$



Matrices

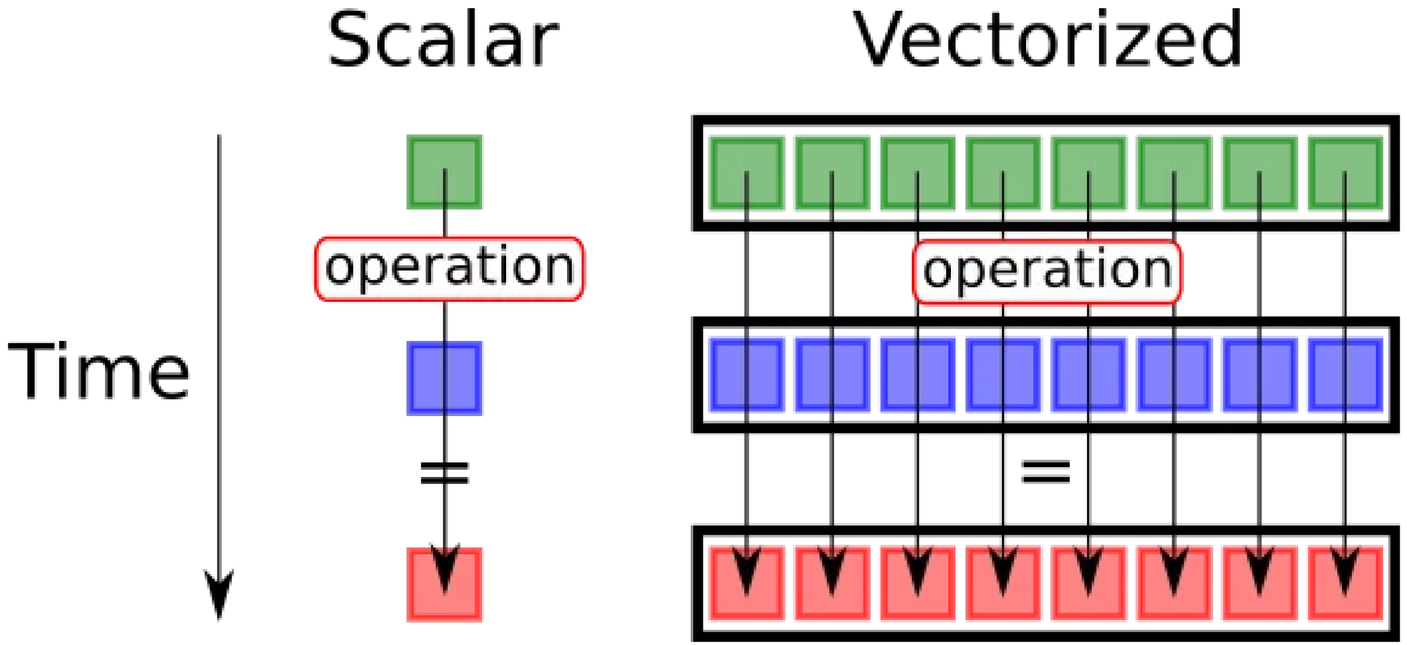
Una matriz es un arreglo bidimensional de valores numéricos, cada elemento es identificado mediante dos indices.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$



Vectorización

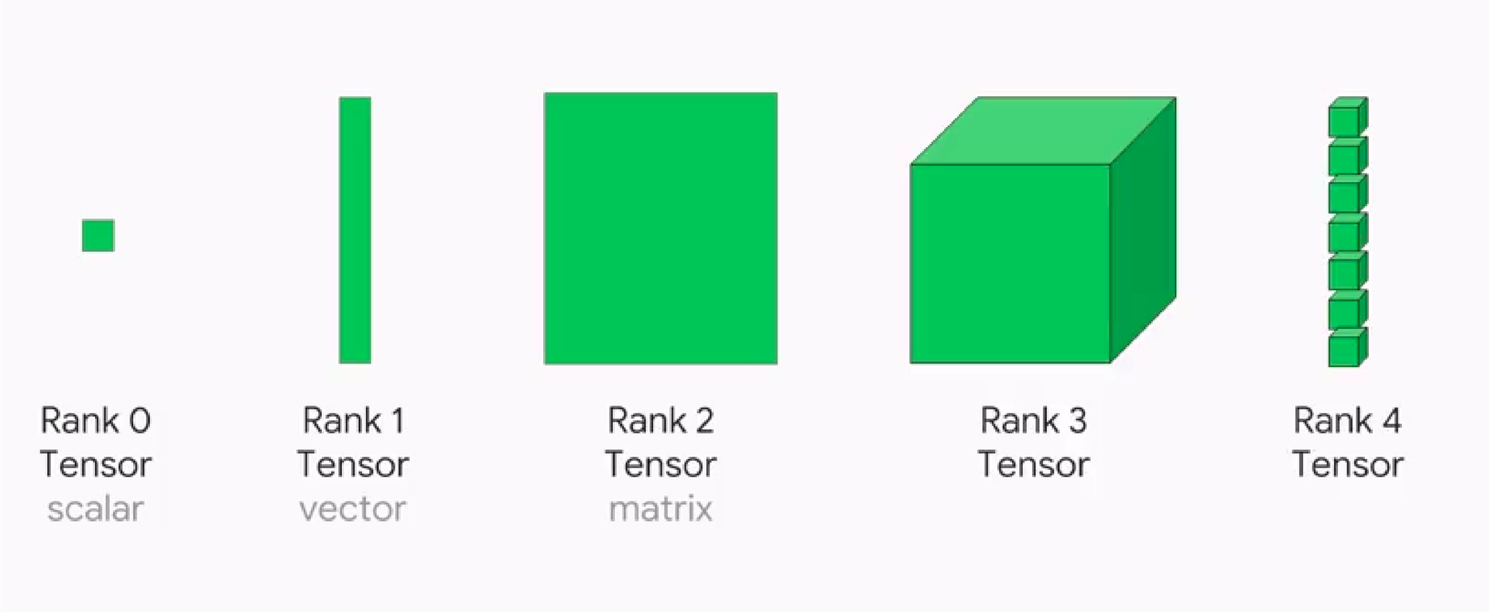
La vectorización es el proceso de convertir un algoritmo que opera sobre un solo valor a la vez a operar en un conjunto de valores a la vez.



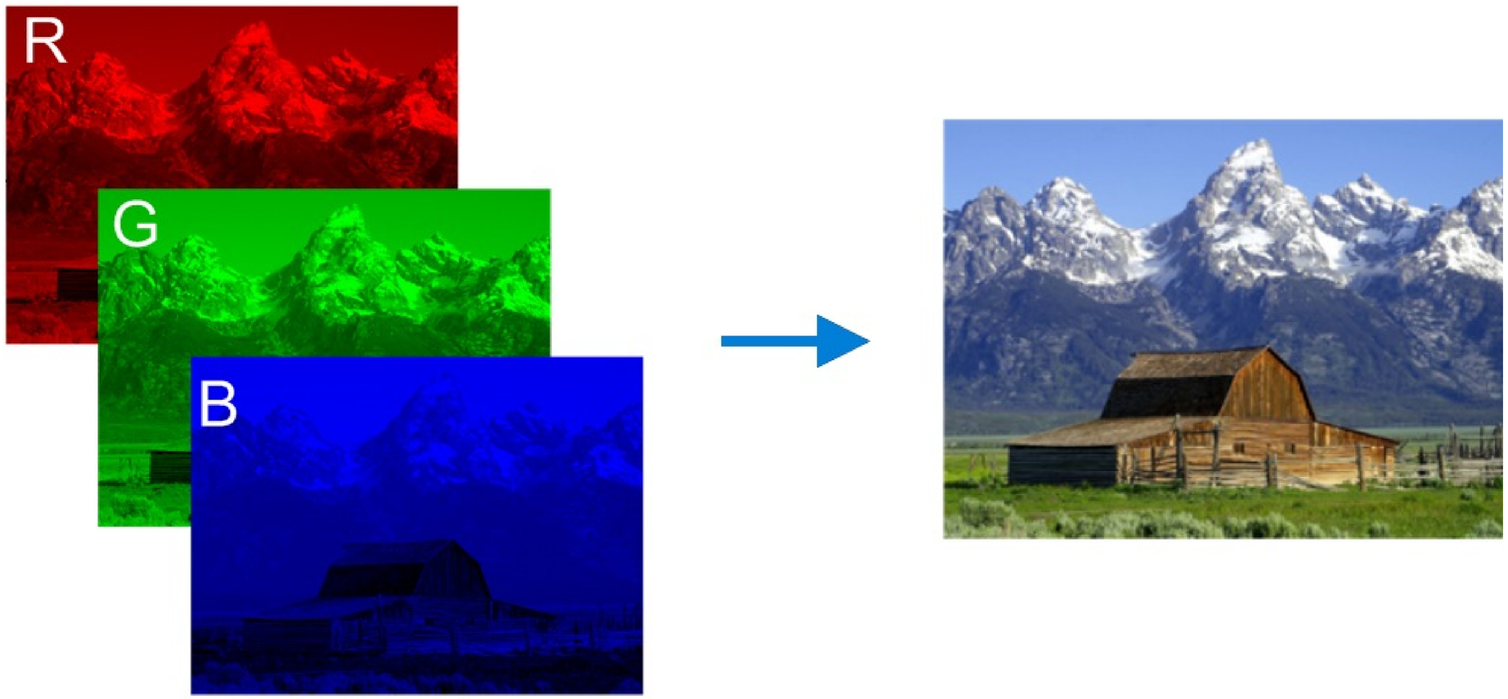
```
import time
w = np.random.rand(1000000)
x = np.random.rand(1000000)
tic = time.time()
c = np.dot(w,x)

toc = time.time()
print('Vectorized version: '+ str(1000*(toc-tic))+'ms')
c = 0

tic = time.time()
for i in range(1000000):
    c +=w[i]*x[i]

toc = time.time()
print("For loop: ",str(1000*(toc-tic))+'ms')
```

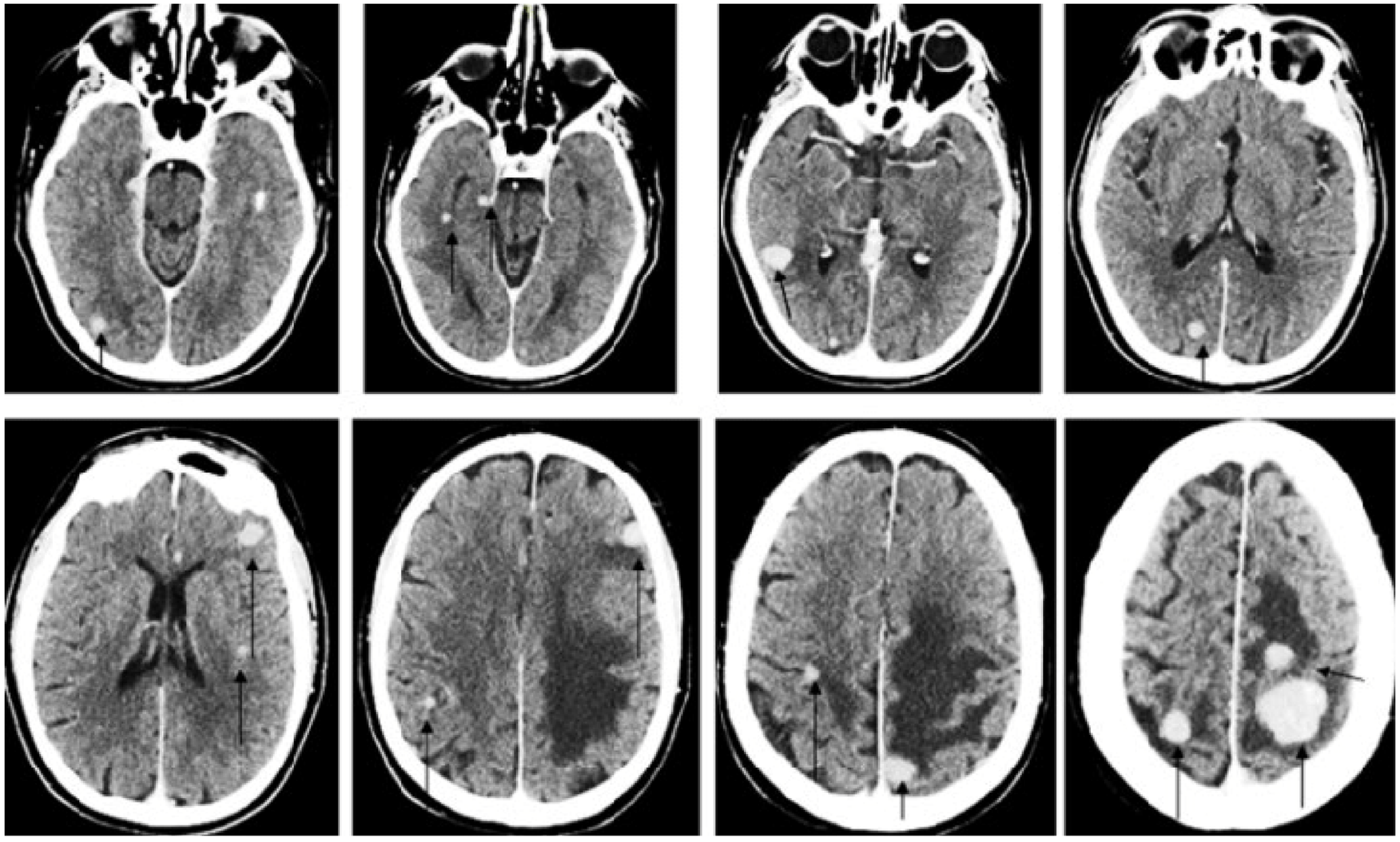
Tensores

En aprendizaje profundo los tensores se emplean como estructura de datos, estos permiten generalizar matrices de más de dos dimensiones, podemos verlos como arreglos multidimensionales.

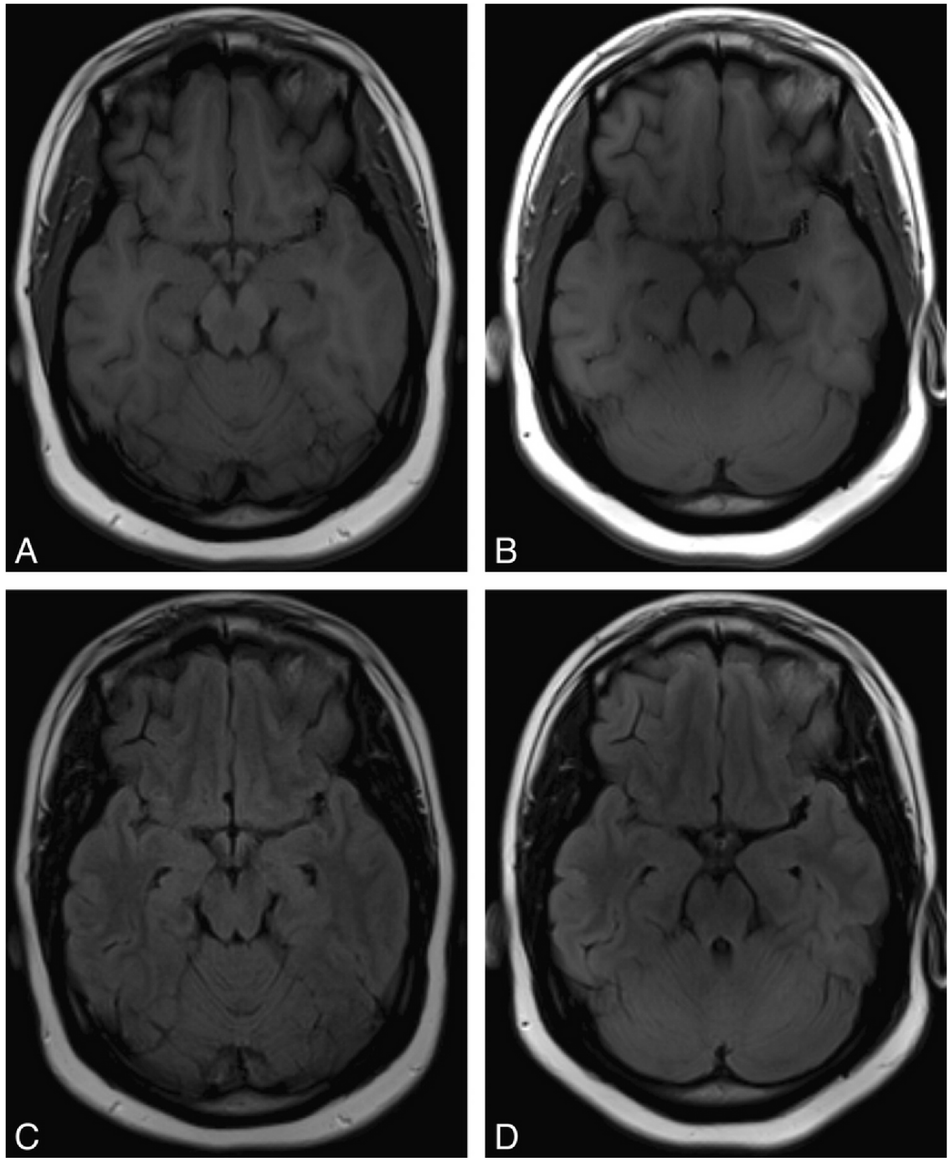


Ejemplos de tensores de rango 3

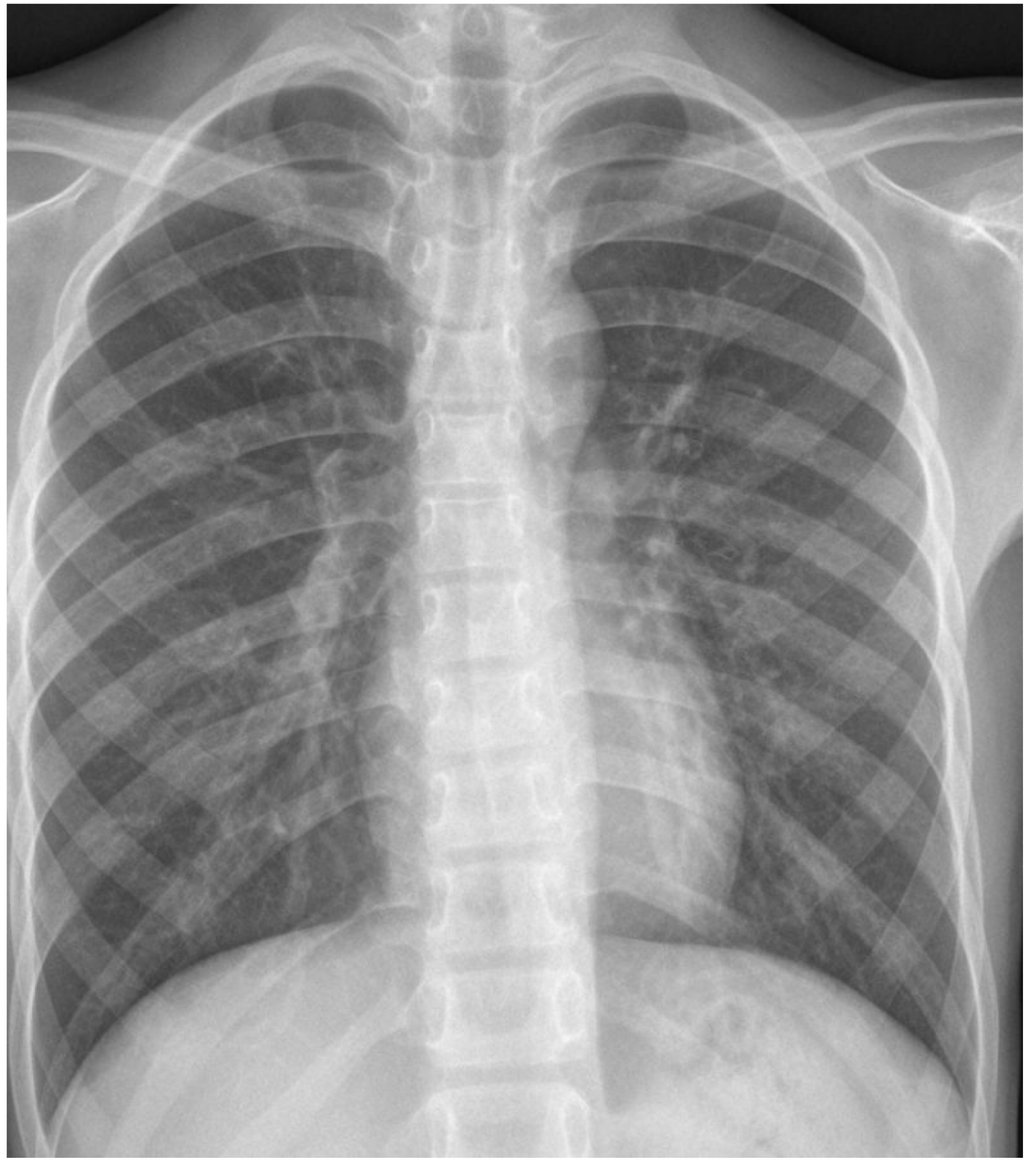




Computerized Tomography

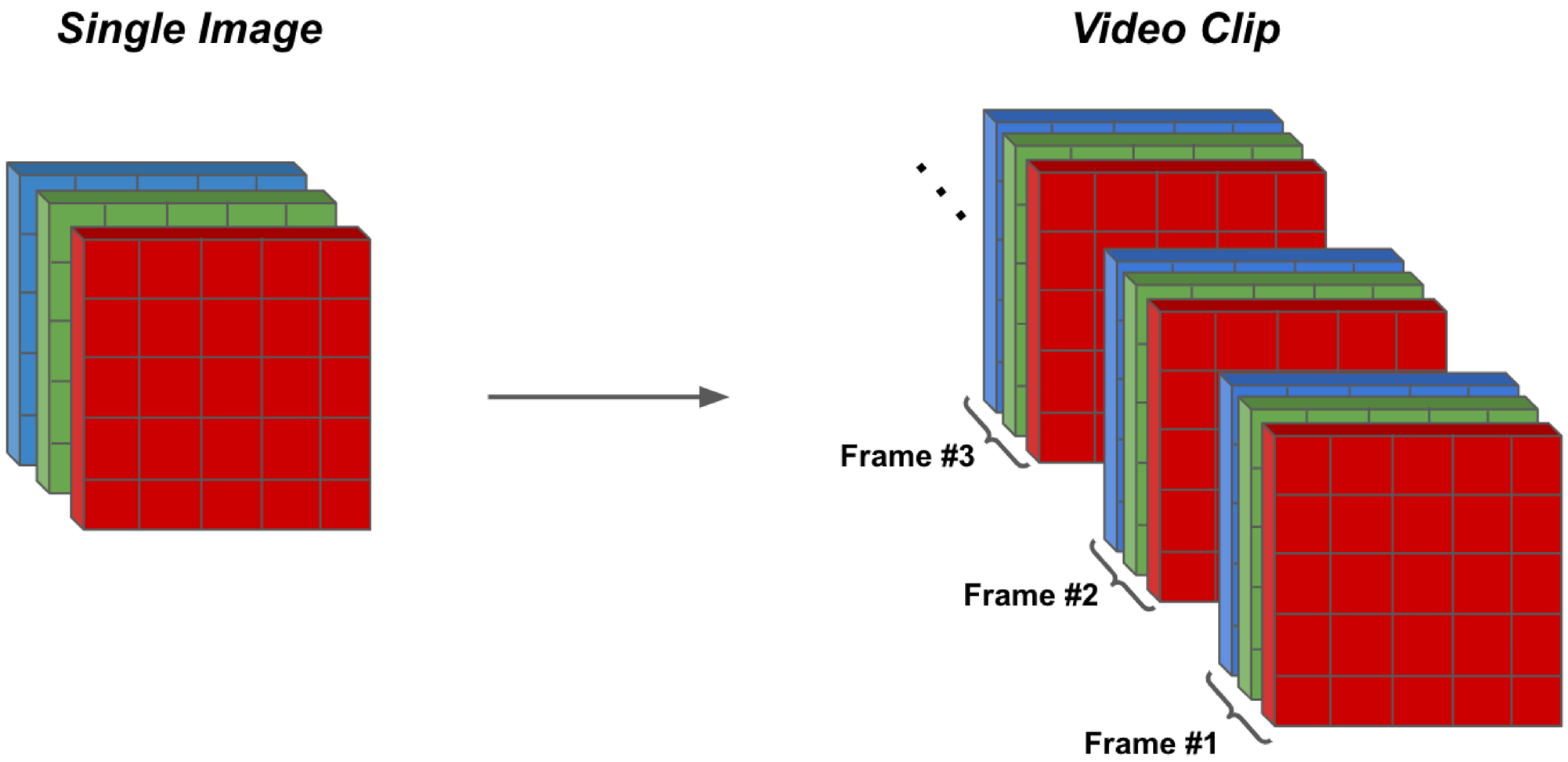


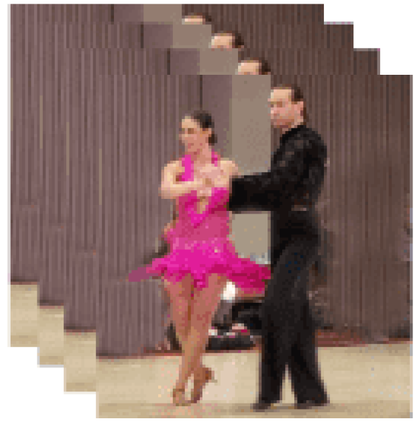
Magnetic Resonance Imaging

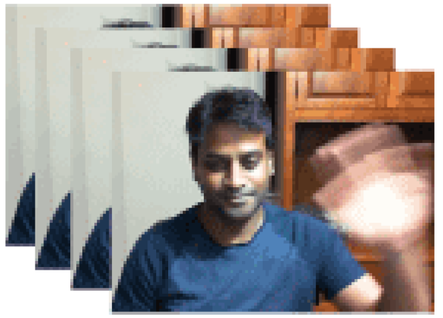


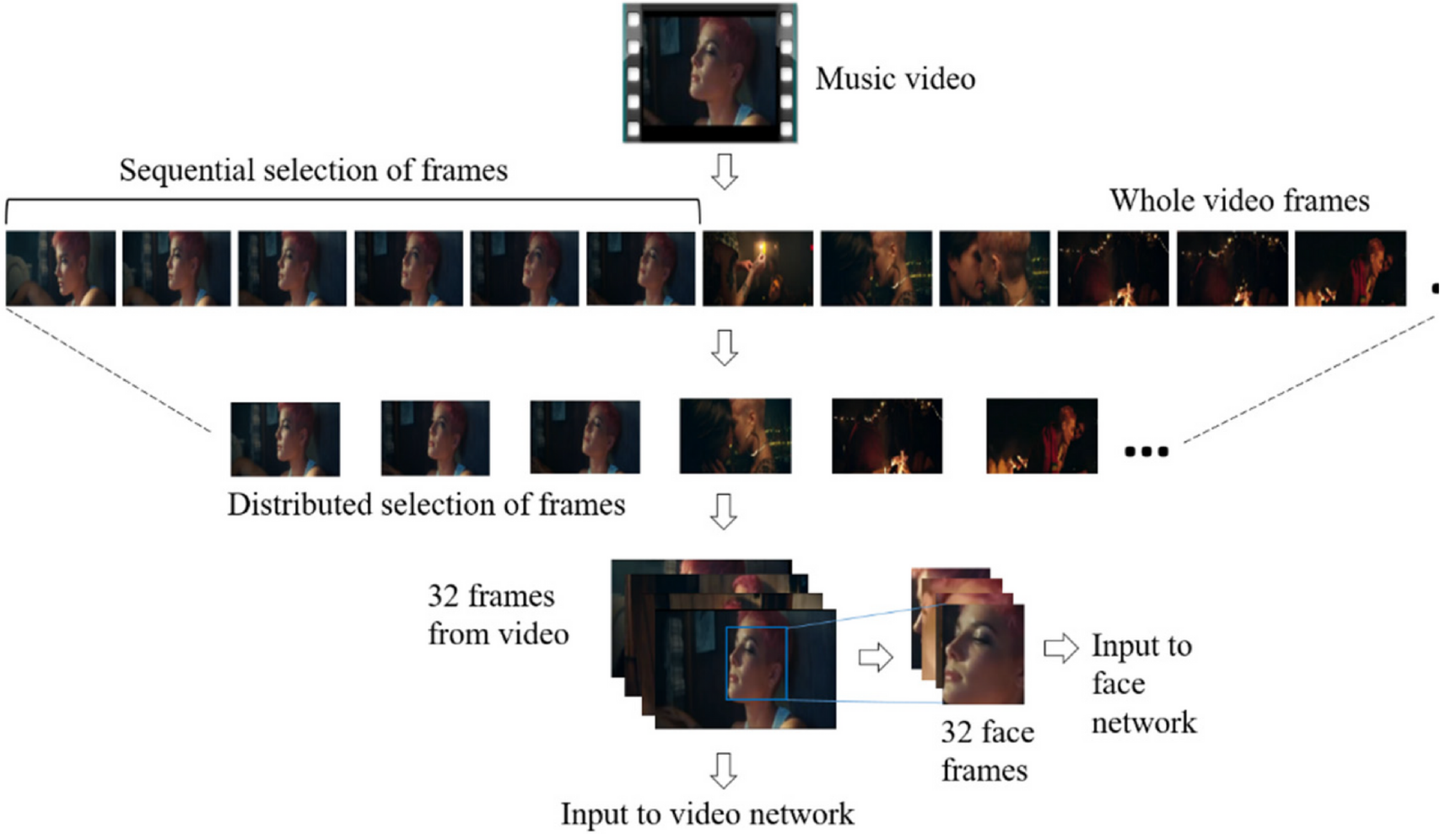
X-ray Imaging

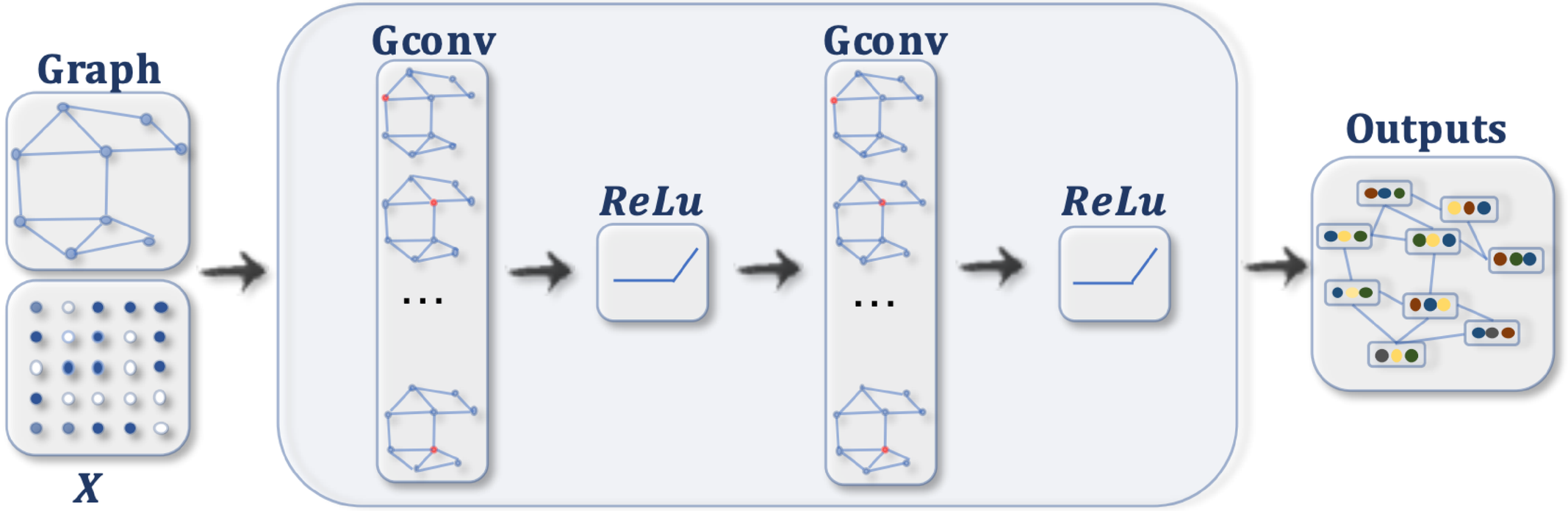
Ejemplos de tensores 4D







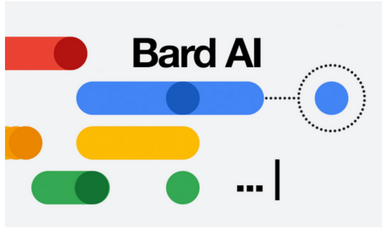

Music video
Sequential selection of frames
Whole video frames
Distributed selection of frames
32 frames
from video
32 face
frames
Input to
face
network
Input to video network


Graph
X
Gconv
...
ReLu
Gconv
...
ReLu
Outputs

Deep Learning

no limits

“a modern house with windows”



Crea una base de datos llamada "Tienda en línea" que contenga las siguientes tablas:

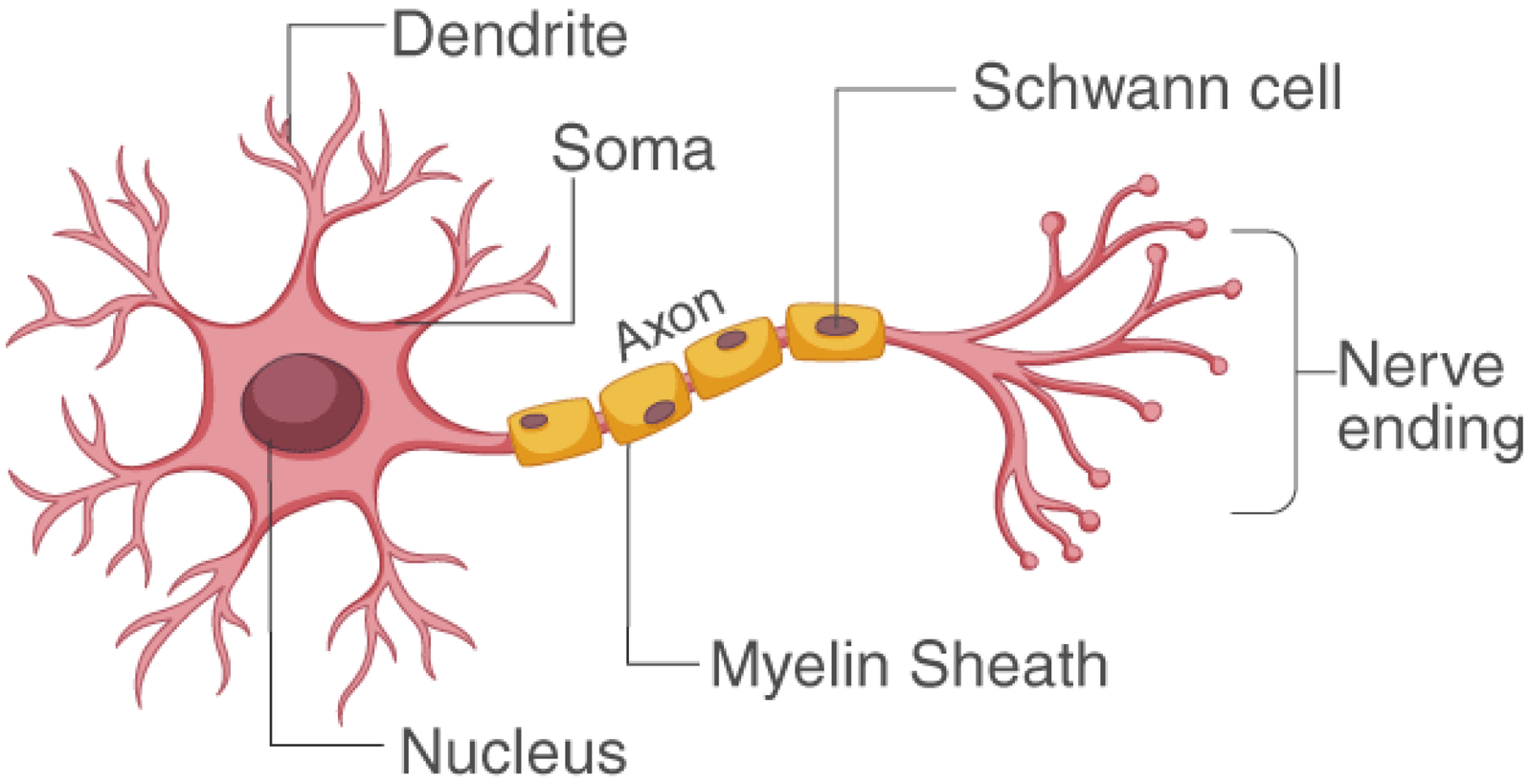
"Productos": con los campos "ID", "Nombre", "Descripción", "Precio", "Stock".
"Clientes": con los campos "ID", "Nombre", "Apellido", "Dirección", "Correo electrónico", "Contraseña".
"Pedidos": con los campos "ID", "Fecha", "ID del cliente", "Total del pedido".

Aquí tienes la creación de la base de datos "Tienda en línea" con las tablas "Productos", "Clientes" y "Pedidos":

sql Copy code

```sql
CREATE DATABASE `Tienda en línea`;

USE `Tienda en línea`;

CREATE TABLE `Productos` (
  `ID` INT(11) NOT NULL AUTO_INCREMENT,
  `Nombre` VARCHAR(255) NOT NULL,
  `Descripción` TEXT NOT NULL,
```

STRUCTURE OF NEURON

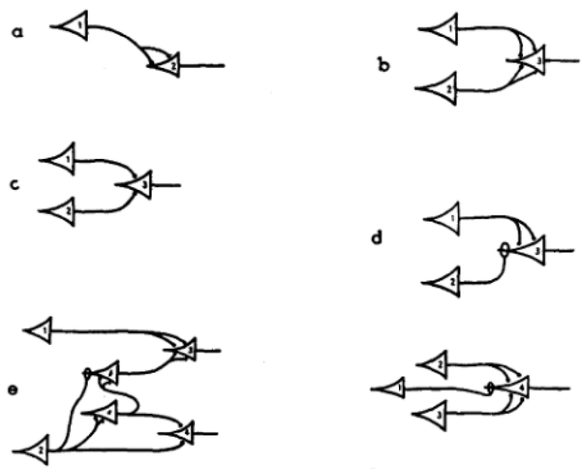


Artificial neural netowrks (ANN) are inspired in biological neural networks

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH and WALTER H. PITTS

originally published in: Bulletin of Mathematical Biophysics, Vol. 5, 1943, p. 115-133



Artificial neural netowrks (ANN) are inspired in biological neural networks

Diagram of a perceptron

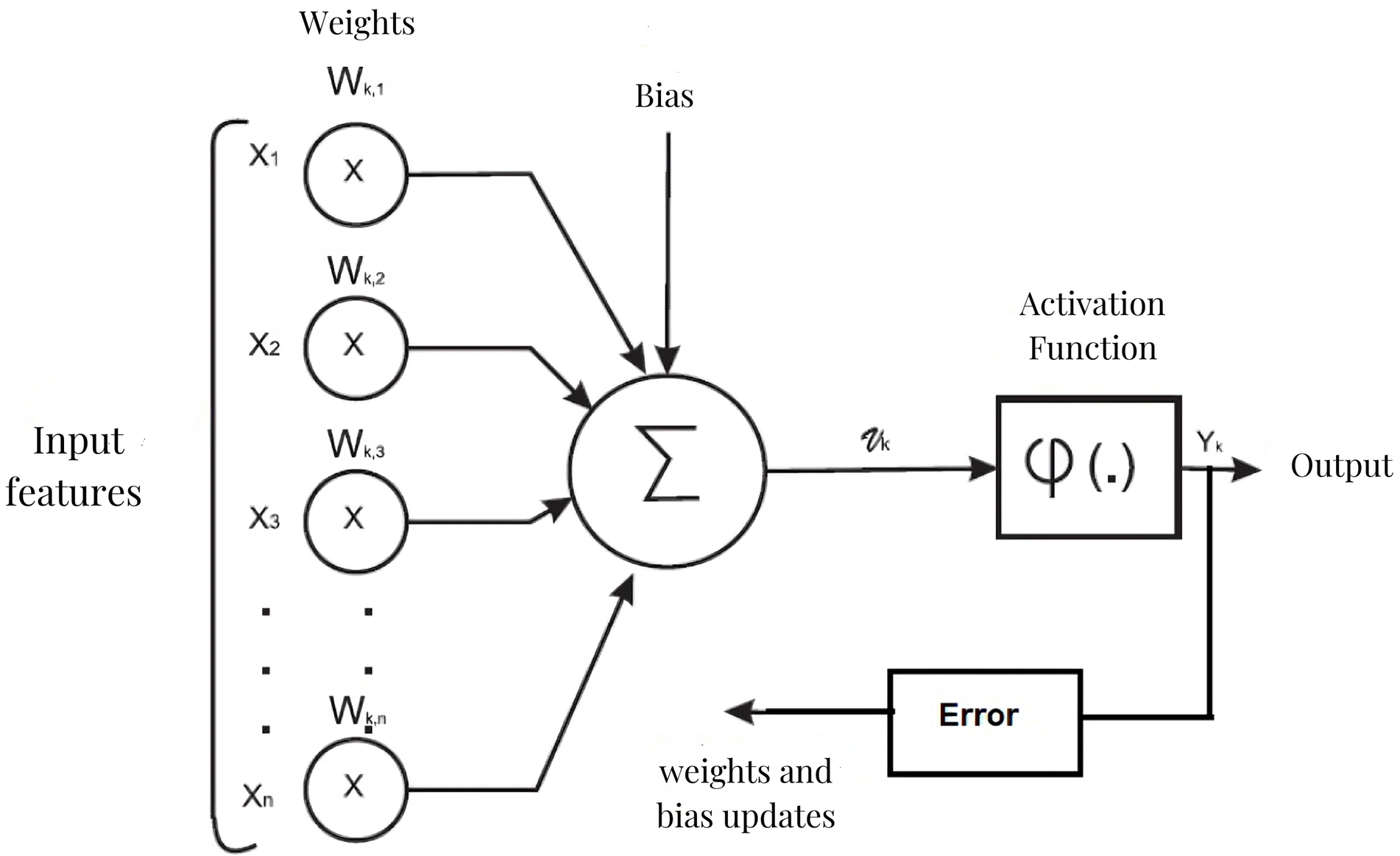
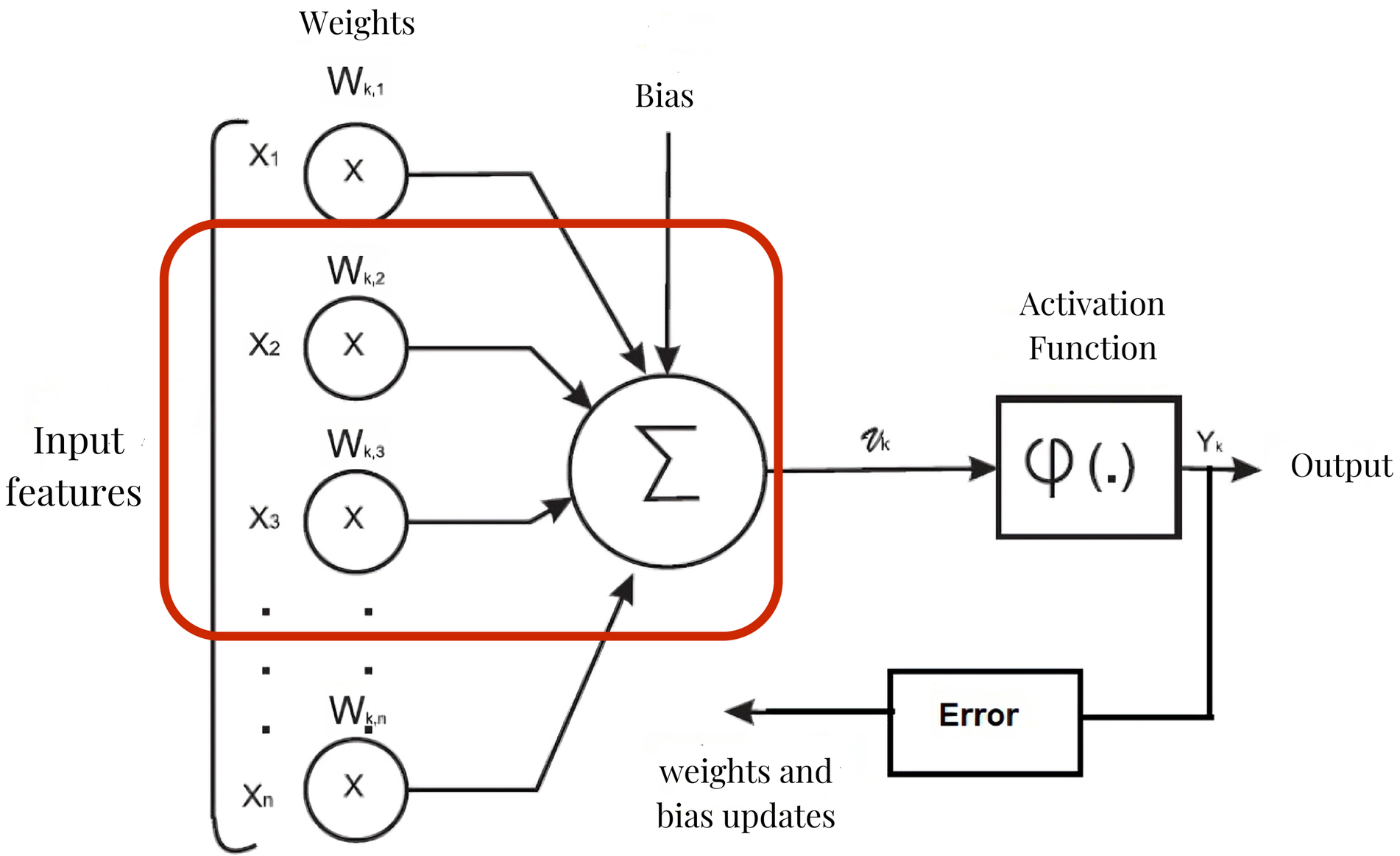
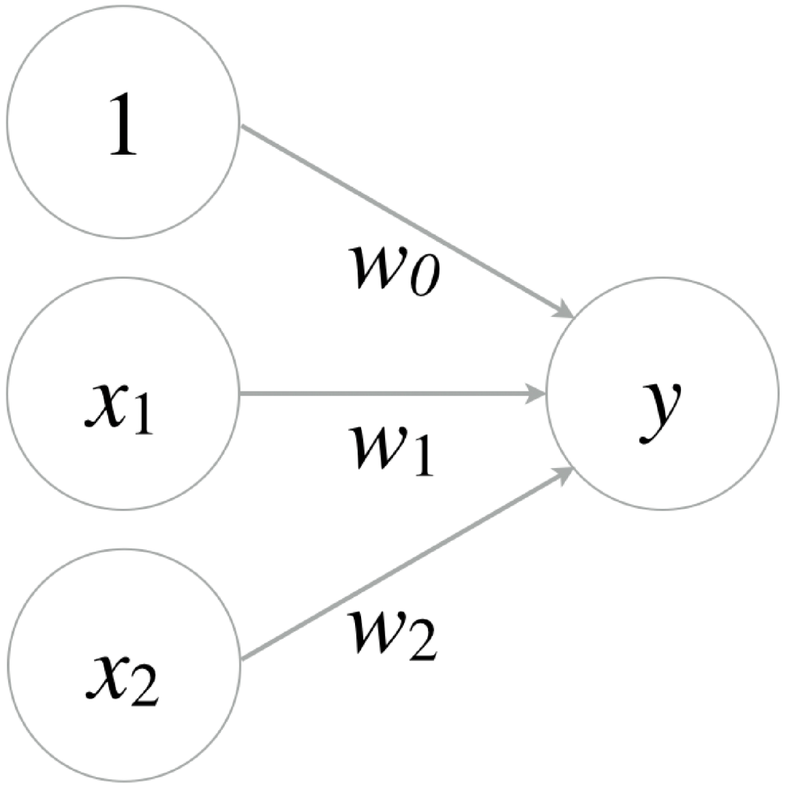


Diagram of a multilayer perceptron

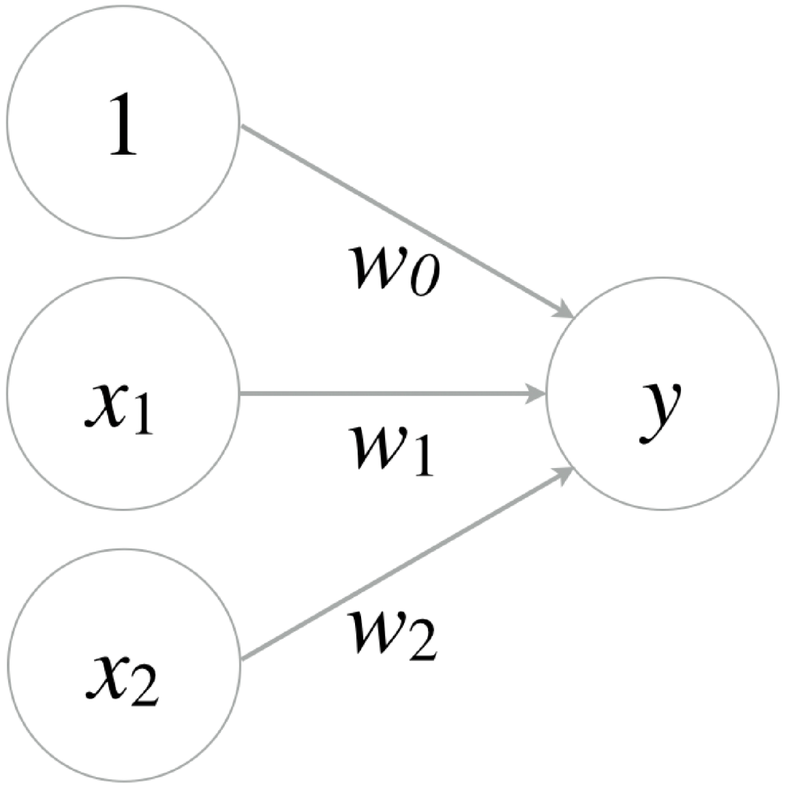


The perceptron



$$y = \sum_{i=0}^{n} x_n \cdot w_n$$

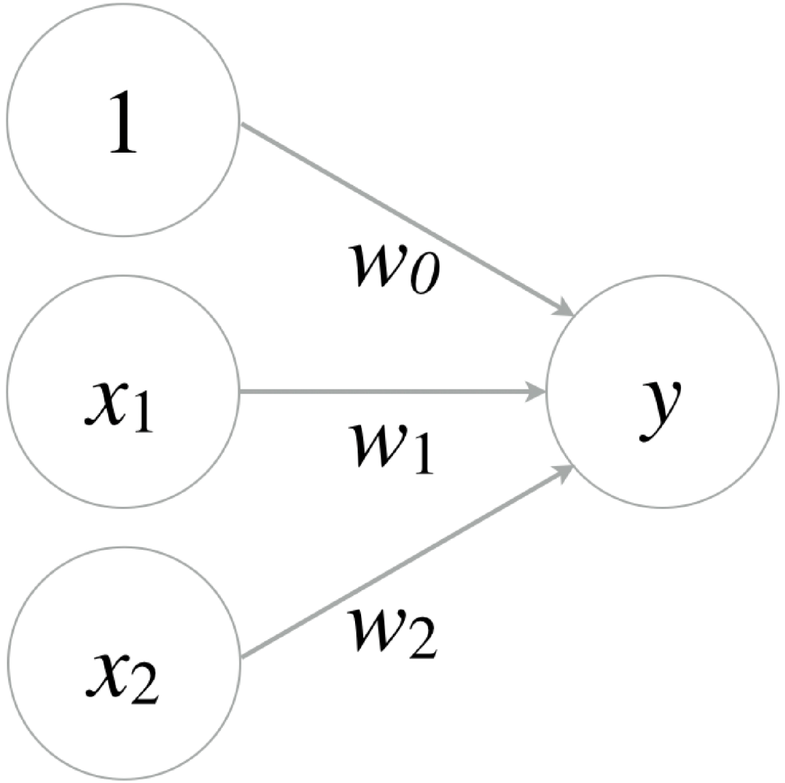
The perceptron



$$y = \sum_{i=0}^{n} x_n \cdot w_n$$

$$\hat{y} = w_0 + \sum_{i=1}^{n} x_n \cdot w_n$$

The perceptron



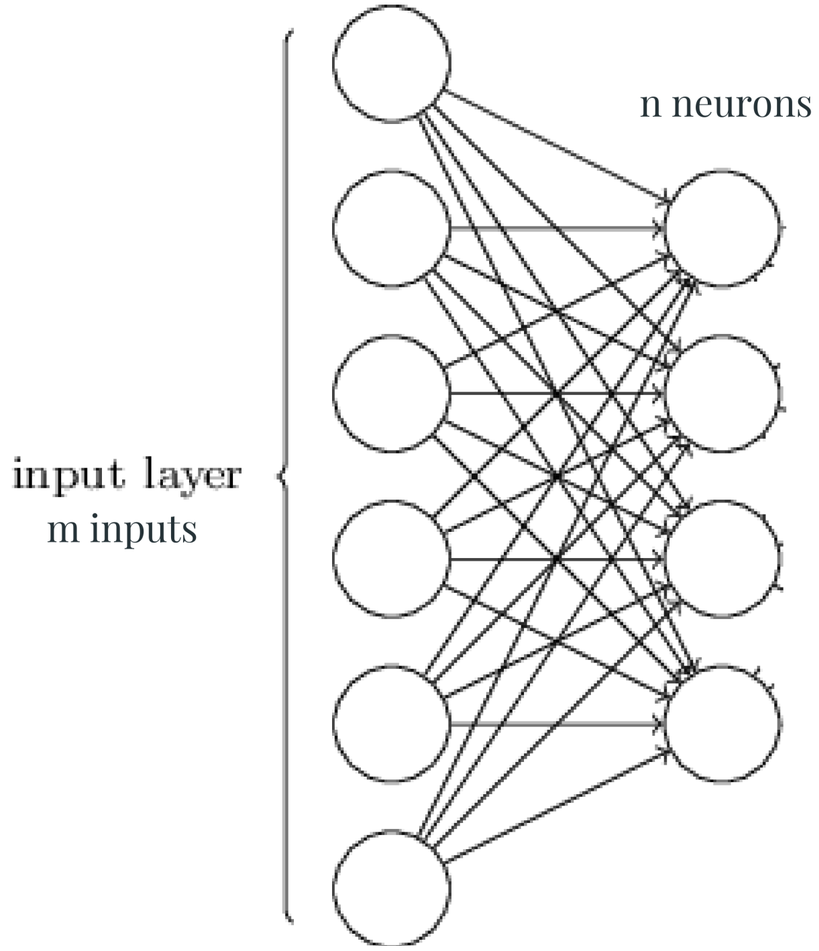
This can be simplified as:

$$\hat{y} = \mathbf{w}^T\mathbf{x} + \mathbf{b}$$

where:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

The perceptron



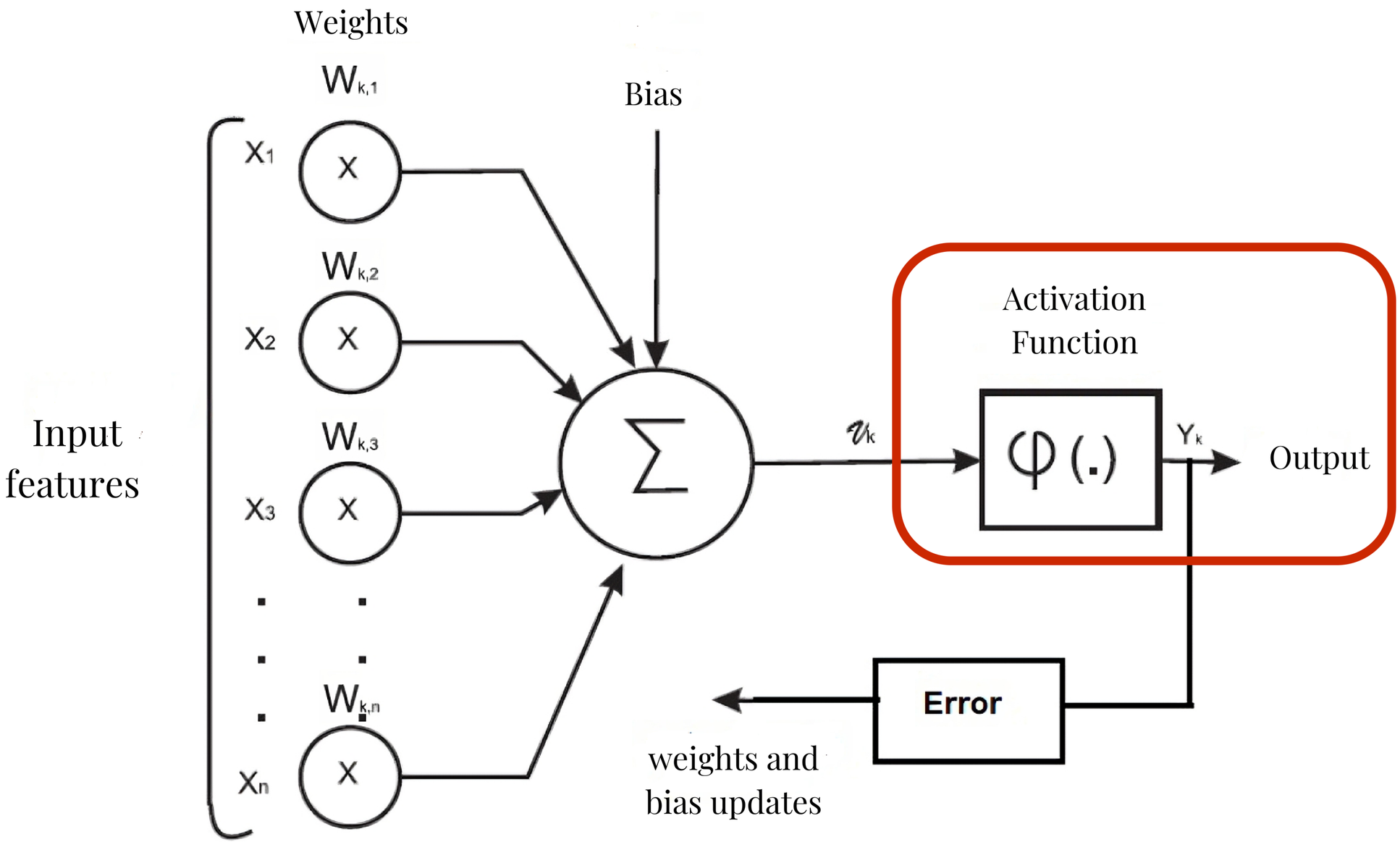
This can be generalized as:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \quad W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1n} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2n} \\ \vdots & & & \vdots & \vdots \\ w_{m1} & w_{m2} & w_{m3} & \cdots & w_{mn} \end{bmatrix}$$

and obtain;

$$\hat{\mathbf{y}} = \mathbf{W}^T\mathbf{x} + \mathbf{b}$$

Diagram of a multilayer perceptron



Activation Functions

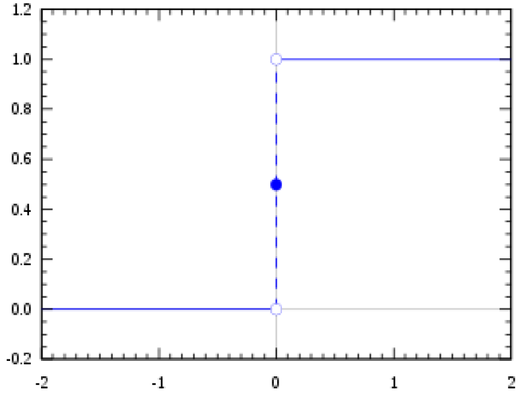
can be expressed as applying a function to the output:

$$\hat{\mathbf{y}} = g(\mathbf{W}^T\mathbf{x} + \mathbf{b})$$

any function can be applied, but a **non-linear function** is expected to be applied

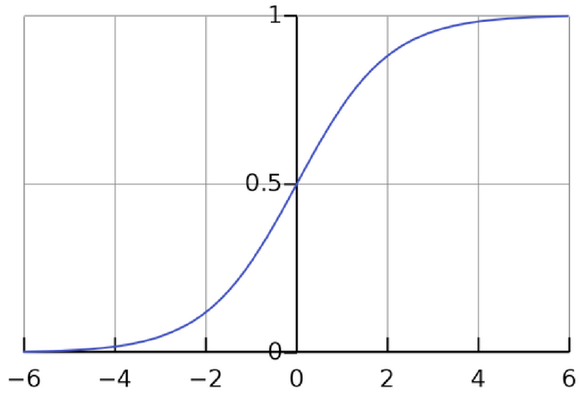
Common Activation Functions

Heaviside



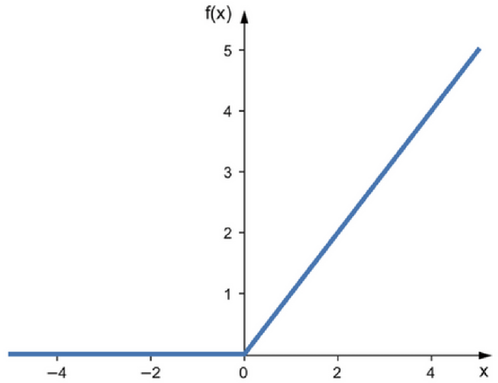
$$g(x) = \begin{cases} 0 & \text{if } x > 0 \\ 1 & \text{if } x \leq 0 \end{cases}$$

Sigmoid



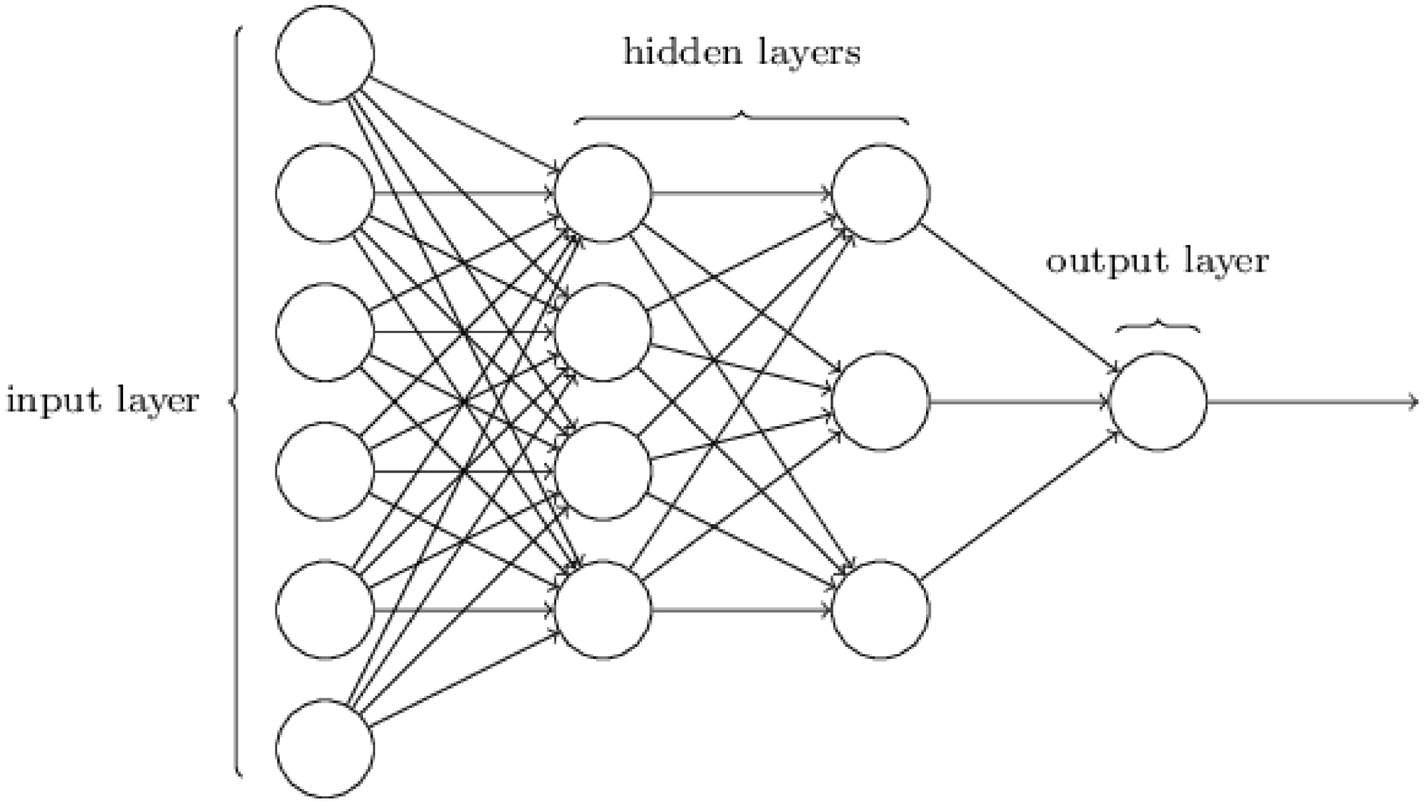
$$g(x) = \frac{1}{1 + e^{-x}}$$

RELU

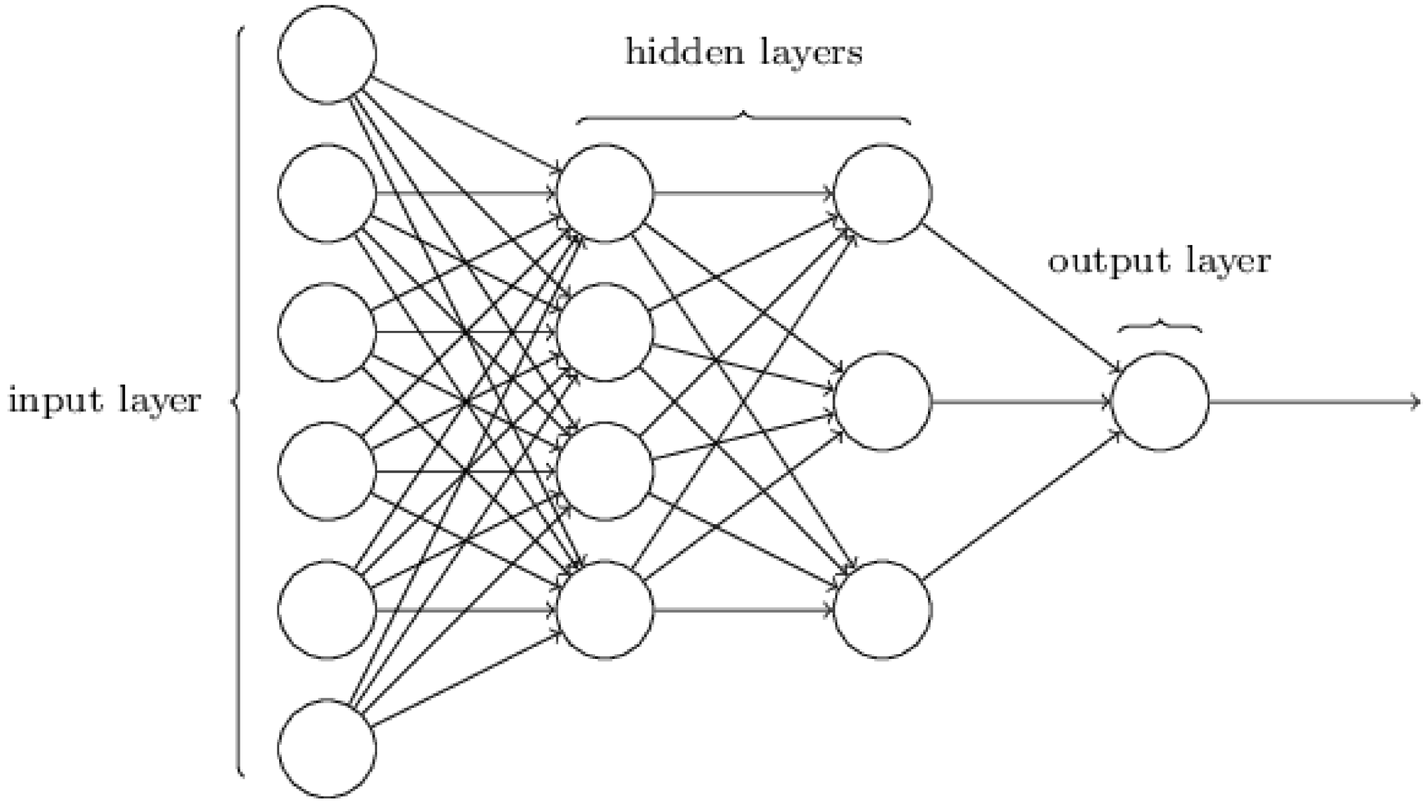


$$g(x) = max(0, x)$$

The perceptron

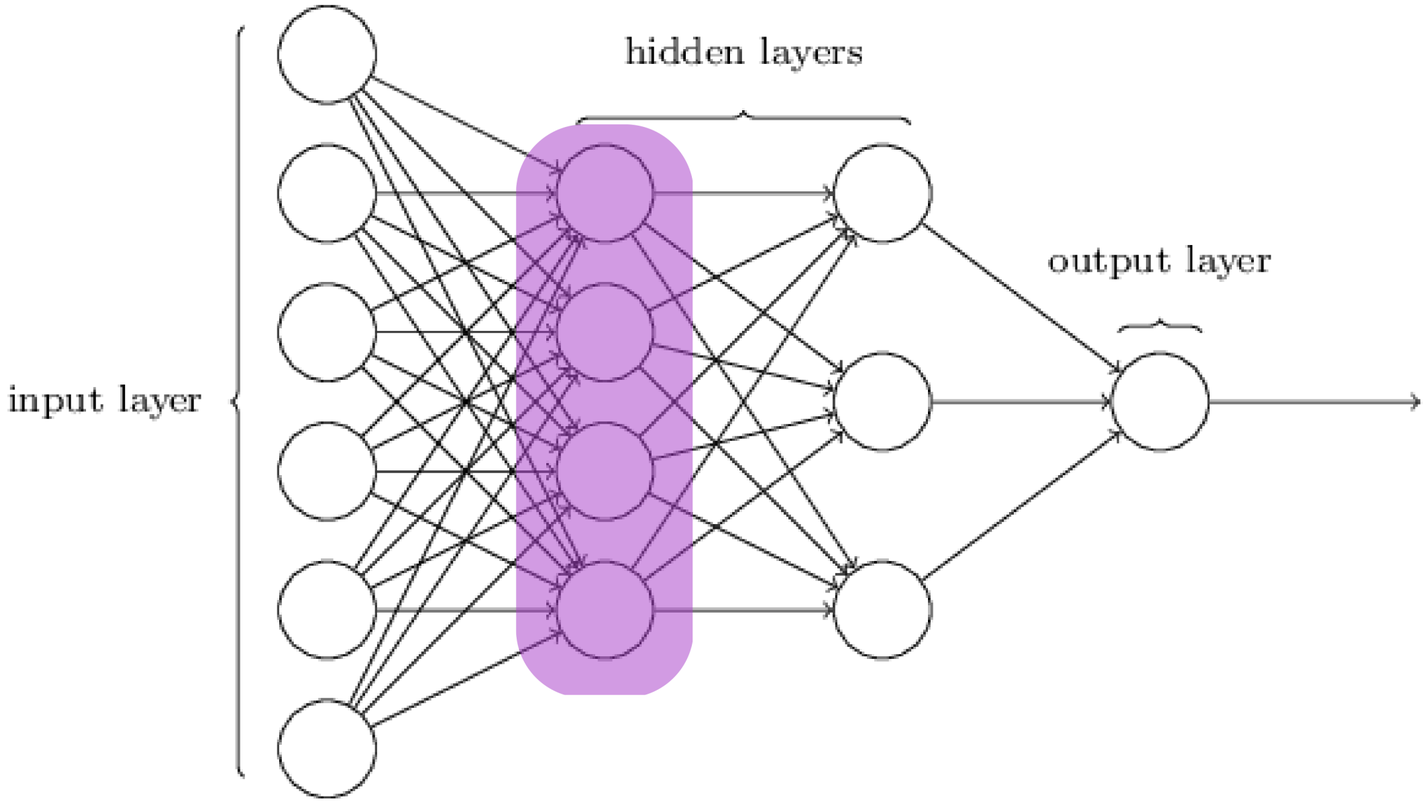


The perceptron



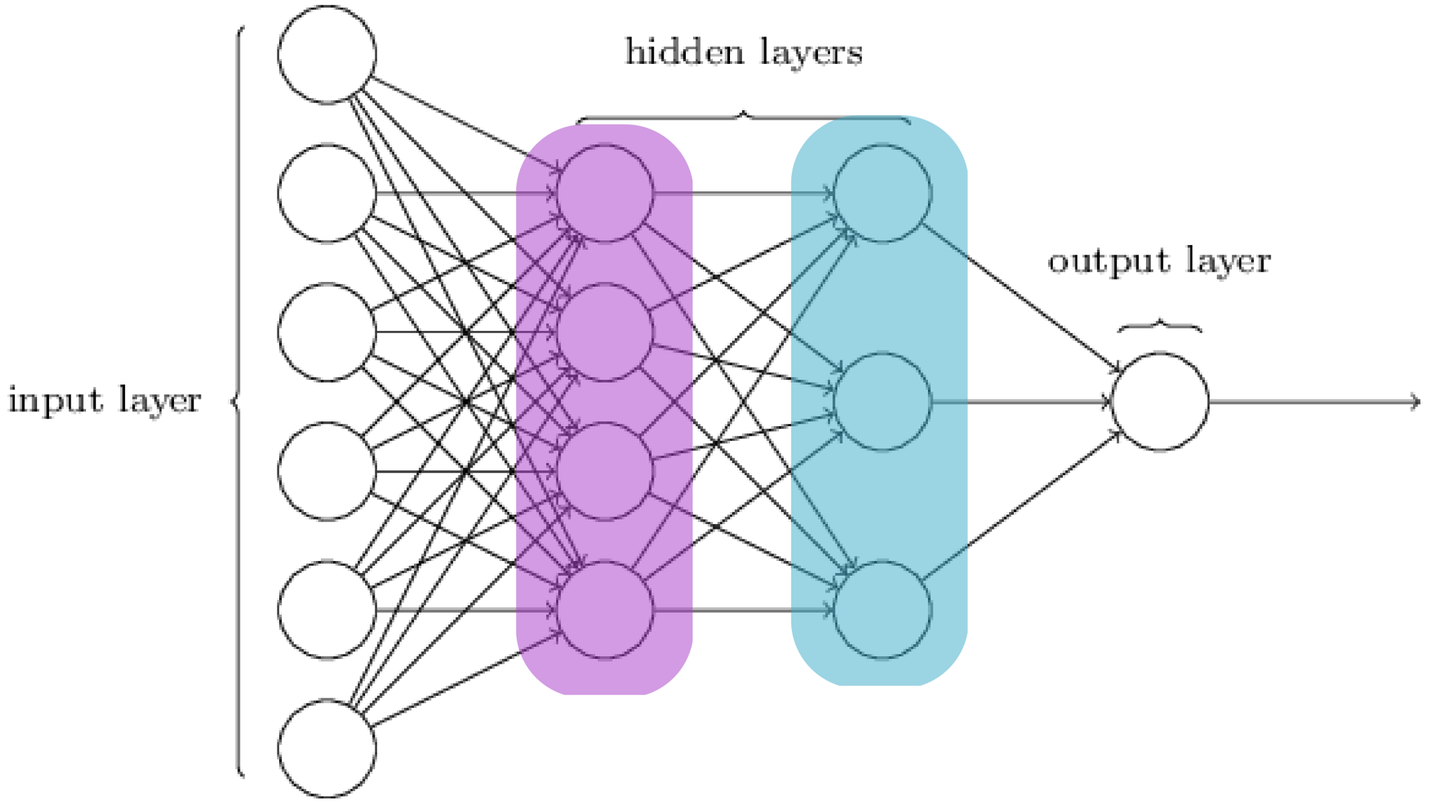
$$\hat{y} = W_3 \cdot g(W_2 \cdot g(W_1 \cdot X + b_1) + b_2) + b_3$$

The perceptron



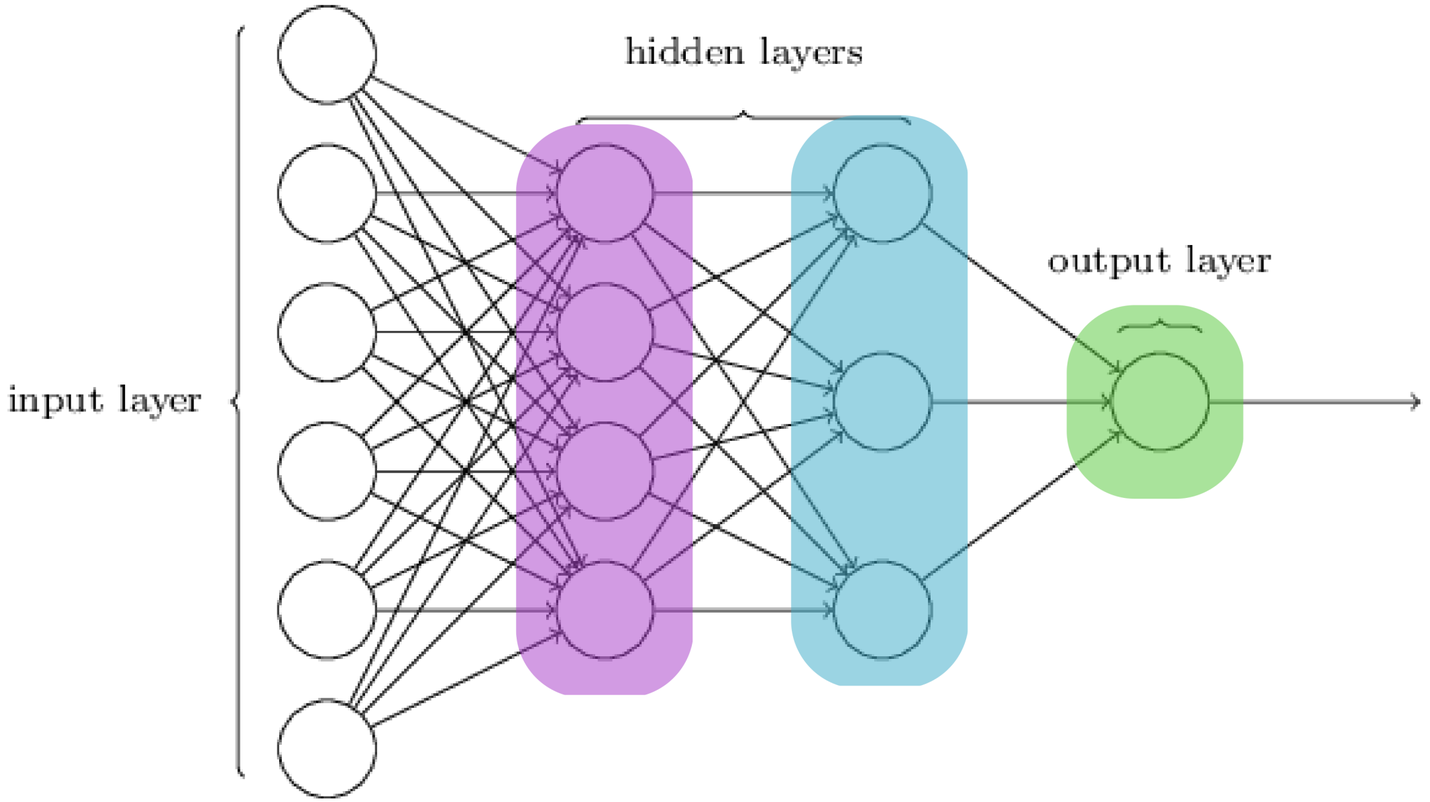
$$\hat{y} = W_3 \cdot g(W_2 \cdot g(W_1 \cdot X + b_1) + b_2) + b_3$$

The perceptron



$$\hat{y} = W_3 \cdot g(W_2 \cdot g(W_1 \cdot X + b_1) + b_2) + b_3$$

The perceptron

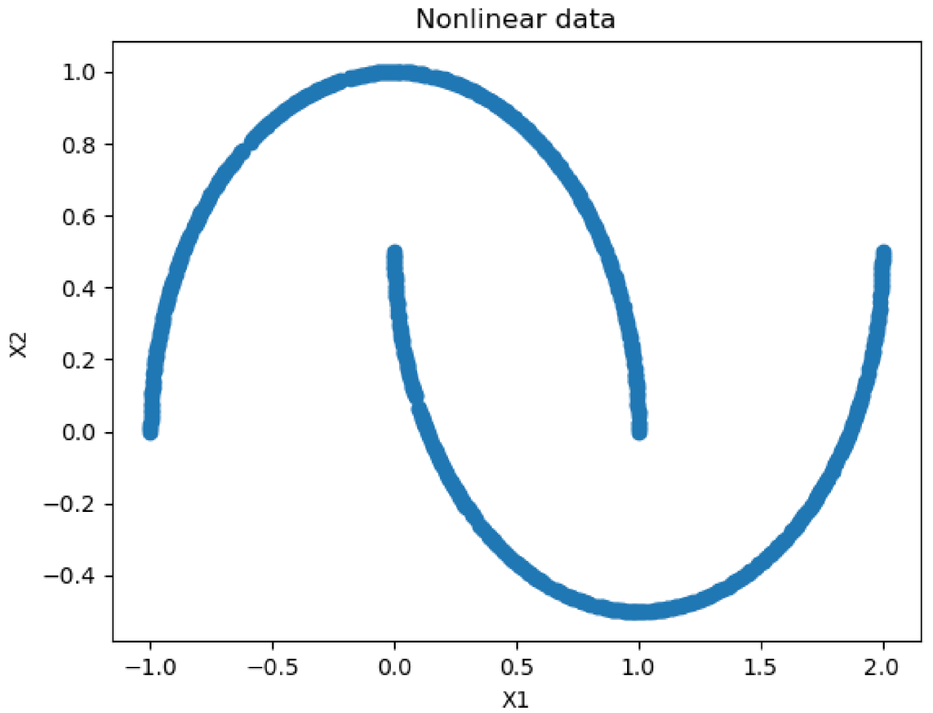


$$\hat{y} = W_3 \cdot g(W_2 \cdot g(W_1 \cdot X + b_1) + b_2) + b_3$$

```
# forward-pass of a 3-layer neural network:

f = lambda x: 1.0/(1.0 + np.exp(-x)) # we will use sigmoid
X = np.random.randn(1,3) # random input vector (3x1)

W1 = np.random.randn(3,8)
W2 = np.random.randn(8,8)
W3 = np.random.randn(8,1)
b1,b2,b3 = 0.7,2.5,9

h1 = f(np.dot(X,W1) + b1)
h2 = f(np.dot(h1,W2) + b2)

out = f(np.dot(h2,W3) + b3)
```

Activation Functions

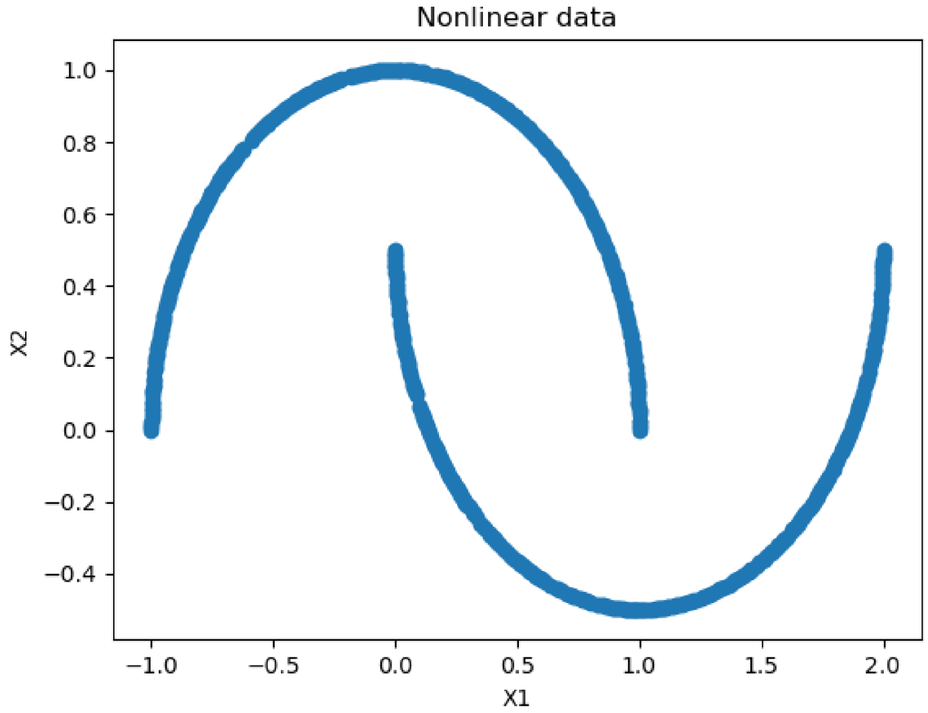
why is non-linearity necessary?

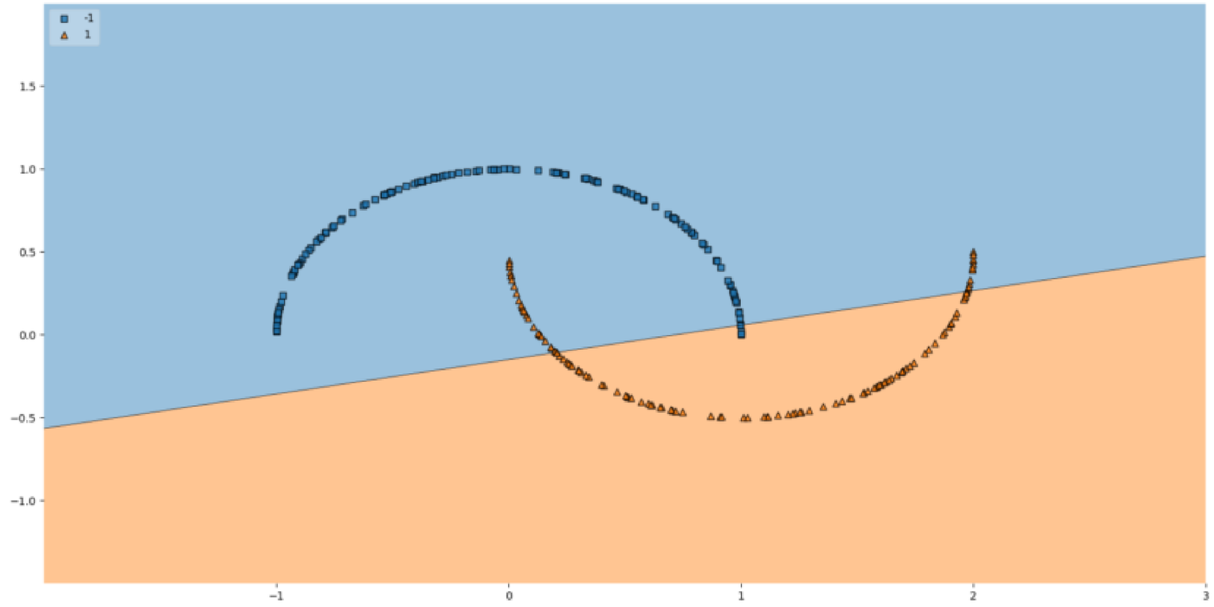


almost all data or problems are nonlinear

Activation Functions

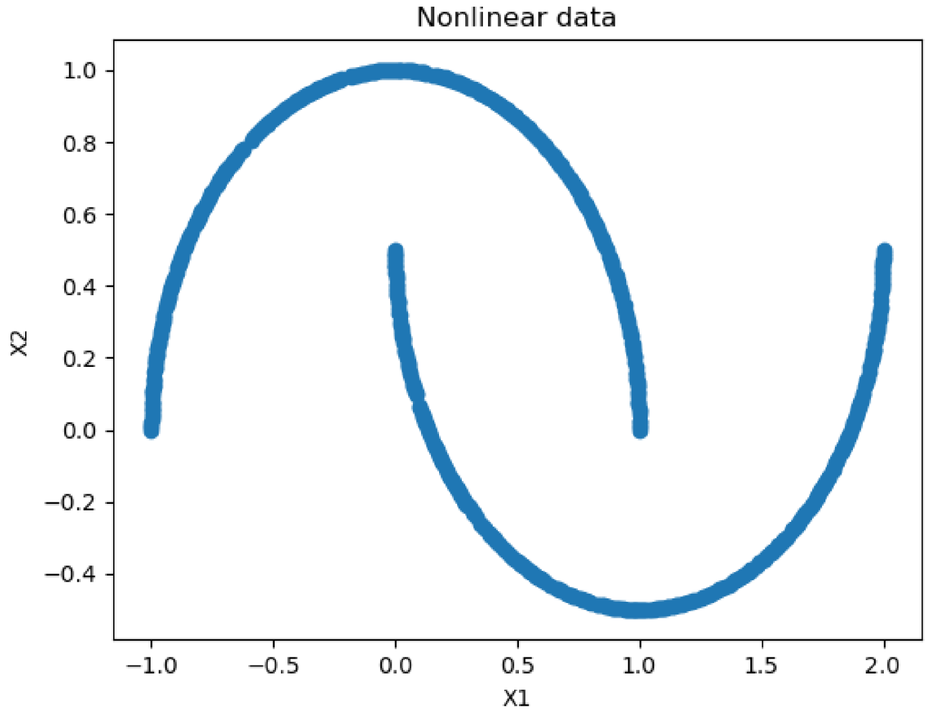
if we do not use an activation function, we can't solve the problem.

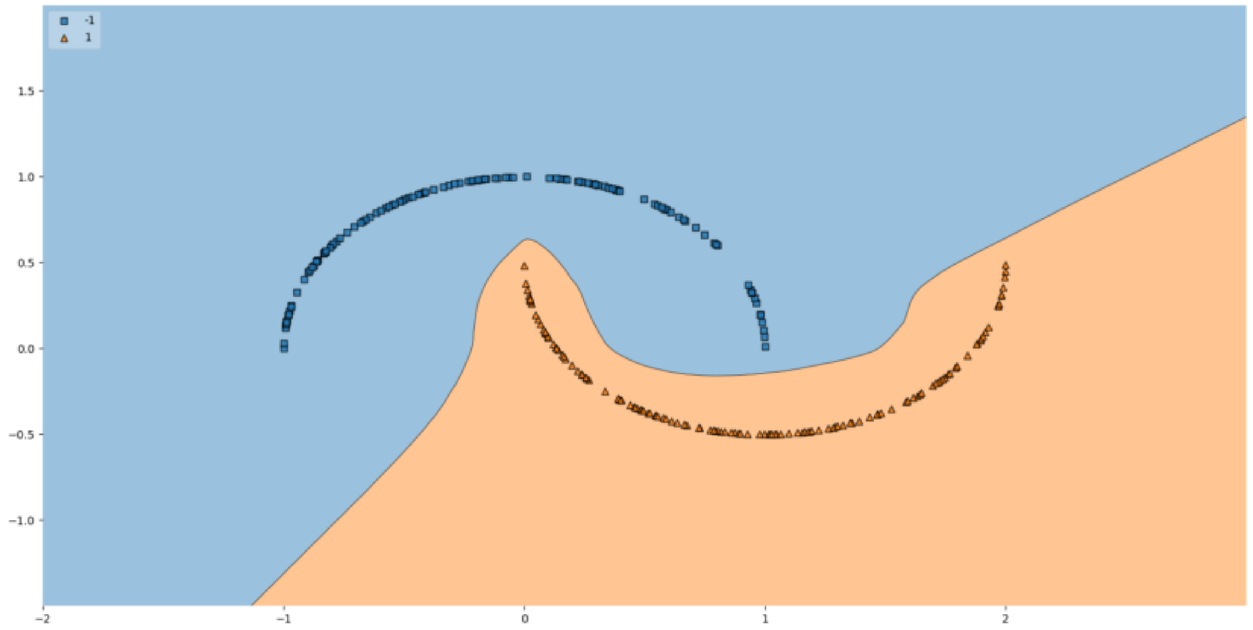




Activation Functions

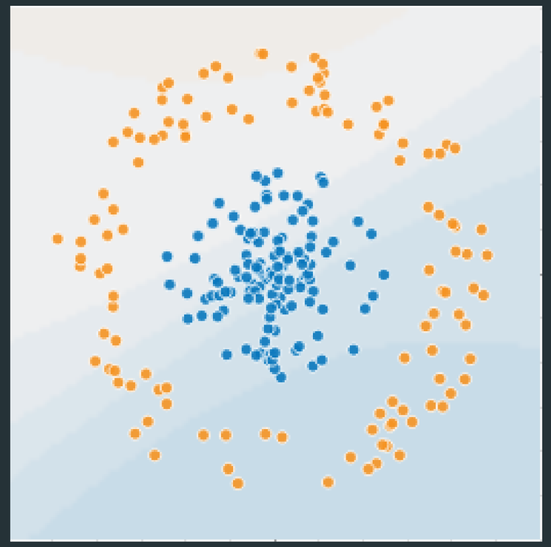
but by introducing a sigmoid it is quite easy to solve it.





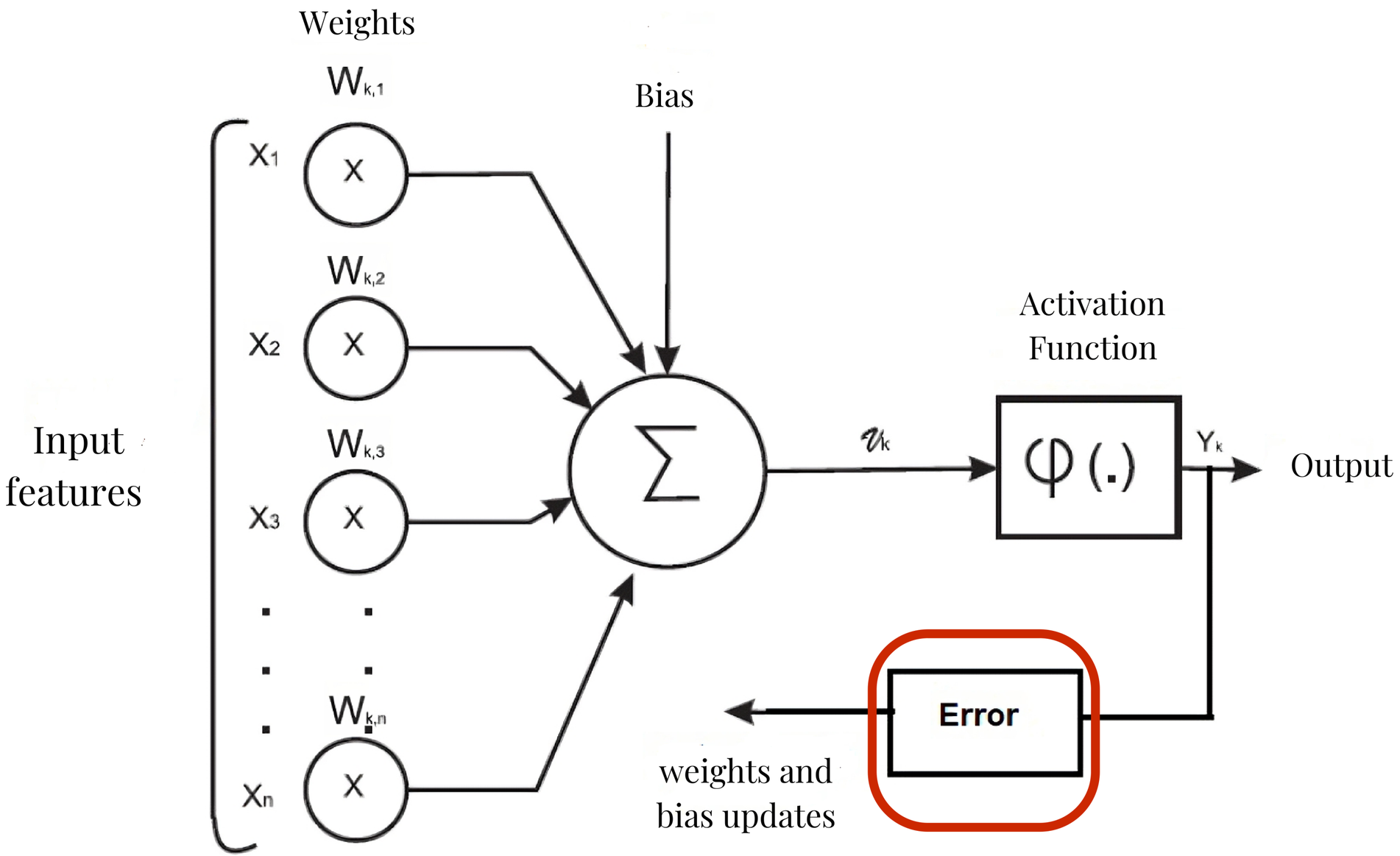
Hands-on

let's solve this problem:



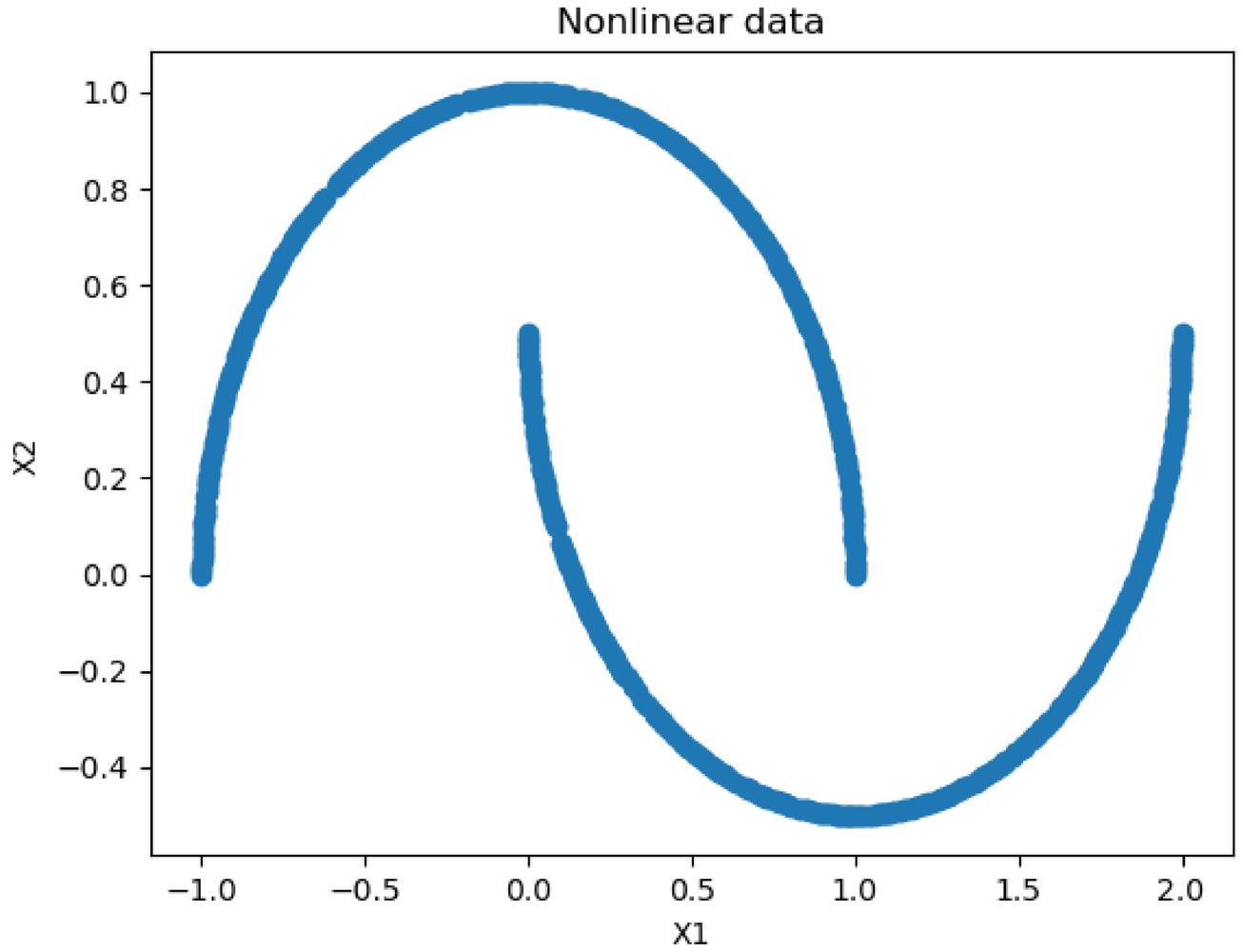
http://playground.tensorflow.org/

Diagram of a multilayer perceptron



Cost Functions

how to quantify the loss?



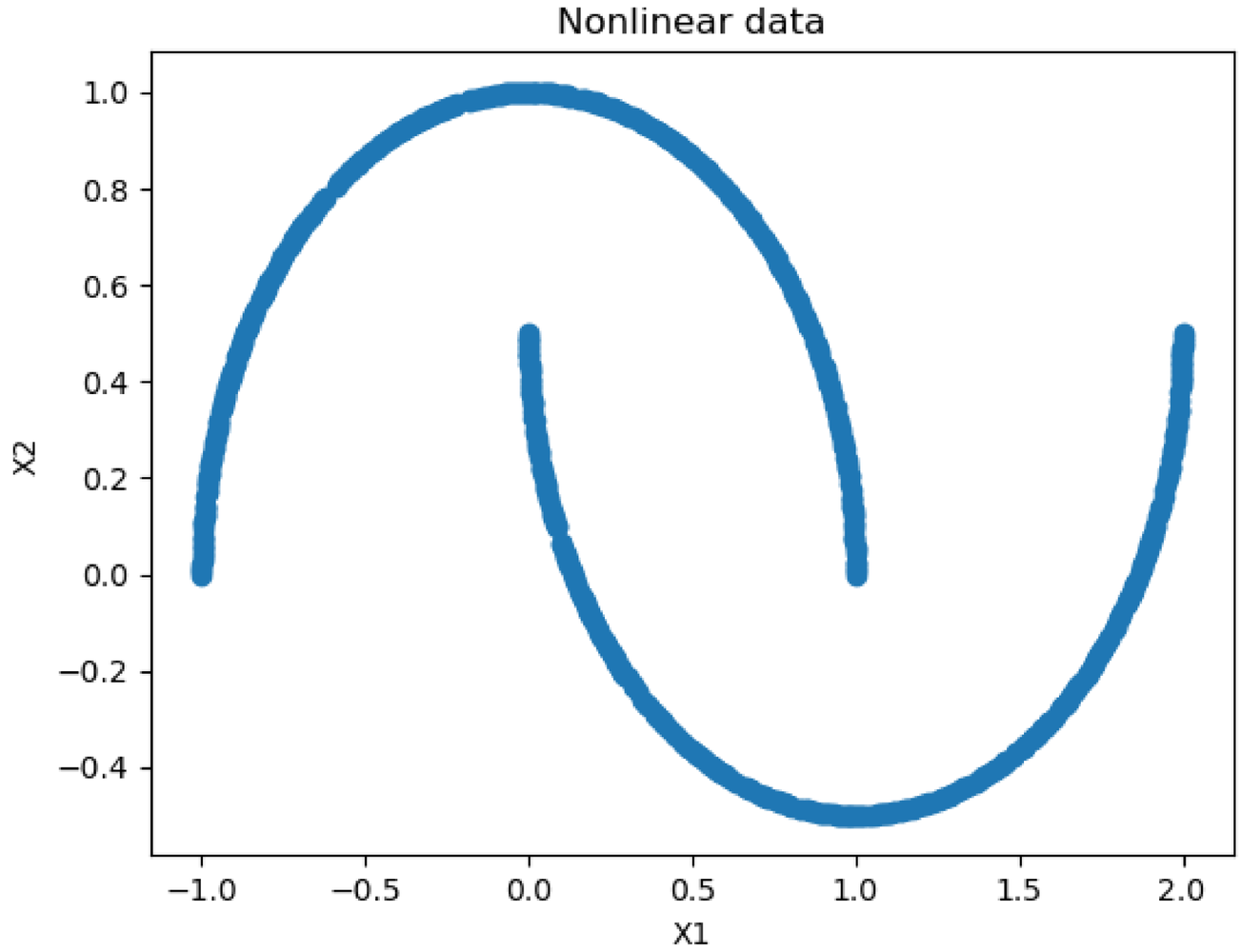
$$x^{(1)} = \begin{bmatrix} 2 \\ 0.4 \end{bmatrix}$$

$$\hat{y} = f(x^{(1)}) = 1$$

$$y = 0$$

Cost Functions

how to quantify the loss?



$$x^{(1)} = \begin{bmatrix} 2 \\ 0.4 \end{bmatrix}$$

$$\hat{y} = f(x^{(1)}) = 1$$

$$y = 0$$

Cost Functions

The loss a function:

$$\mathcal{L}(\hat{y}, y)$$

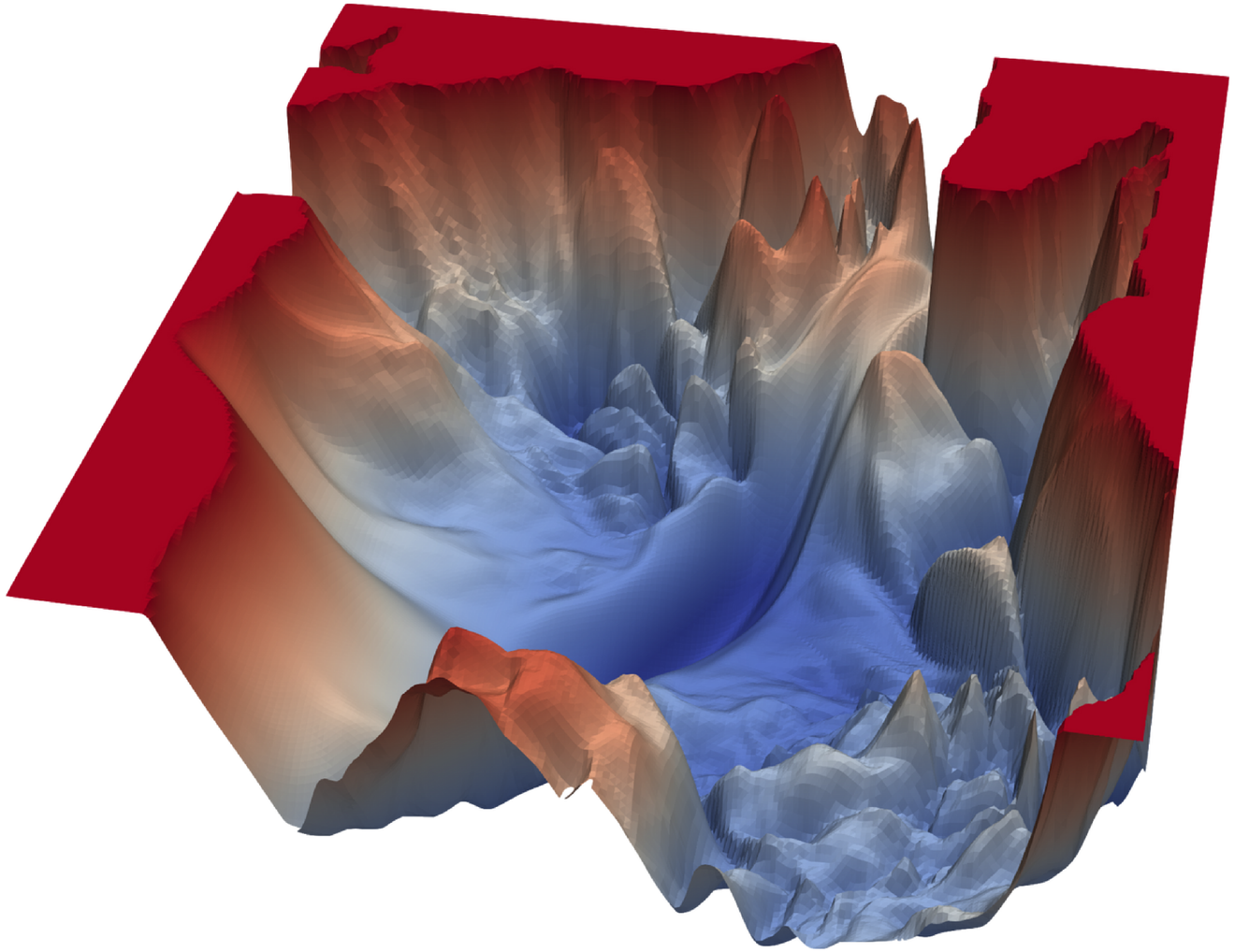
The function must be derivable

For a batch (a subset of data) is expressed as

$$J(W) = \frac{1}{n}\sum_{1}^{n} \mathcal{L}(\hat{y}, y) \longrightarrow$$

Loss function
Cost function
Objective function

Cost Functions



The Cost function of a VGG-56

Common Cost Functions

Binary Cross Entropy

$$-\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)$$

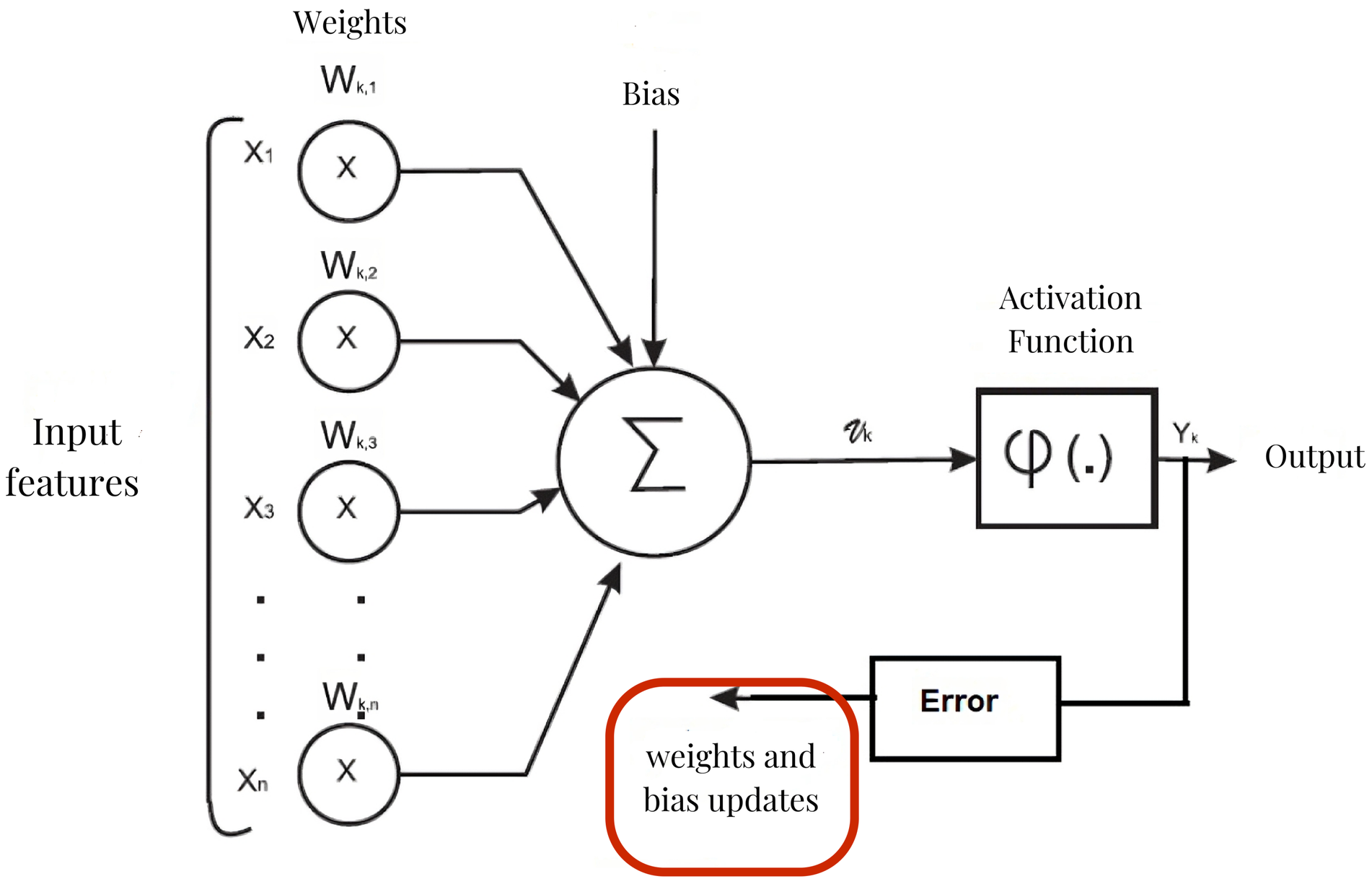
Is used in binary **classifcation**

Mean Squeared Error

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Is used in **regression**

Diagram of a multilayer perceptron



Backpropagation

we want to find the net weights at which that achieve the best performance and the lowest loss

$$W^* = \underset{W}{\operatorname{argmin}} f(W, \hat{y}, y)$$

we want to minimize this

Backpropagation

we want to find the net weights at which that achieve the best performance and the lowest loss

$$W^* = argmin_W \mathcal{L}(W)$$

$$\downarrow$$

we want to minimize this

to find the optimal weights

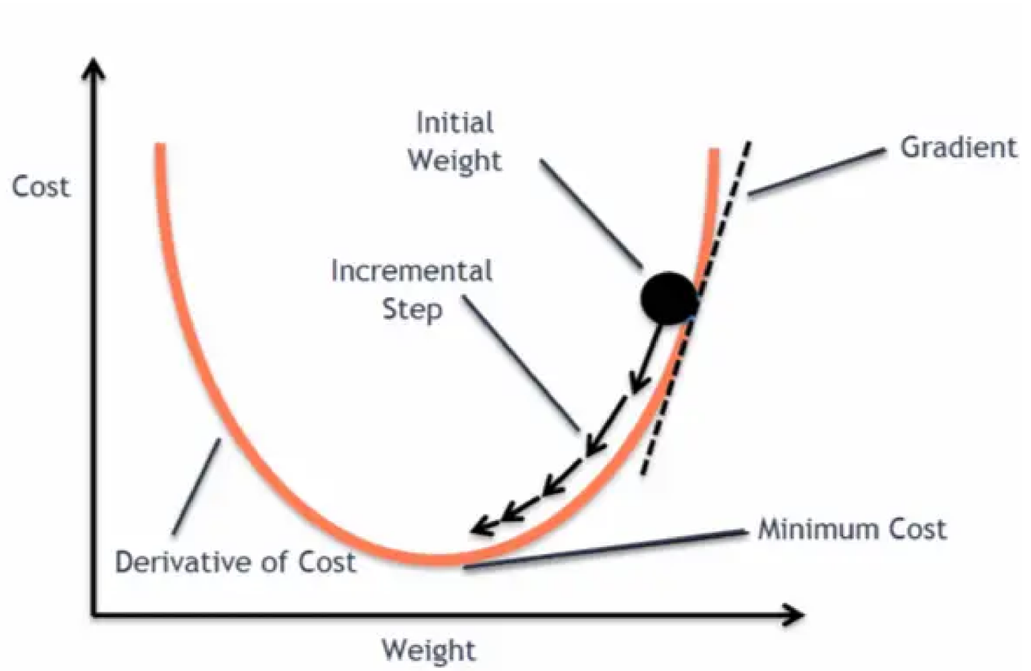
optimizers: sgd

we start with some random weights

and let's find the derivative of

$$\nabla_W = \frac{\partial \mathcal{L}(W)}{\partial W}$$

this is the gradient



optimizers: sgd

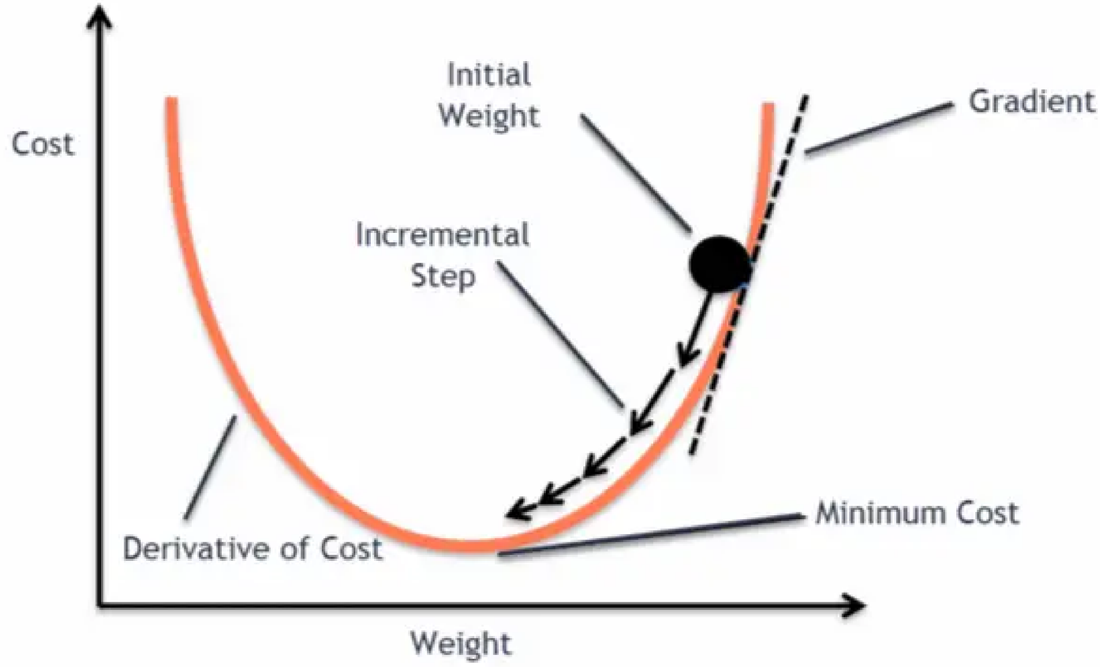
we start with some random weights

and let's find the derivative of

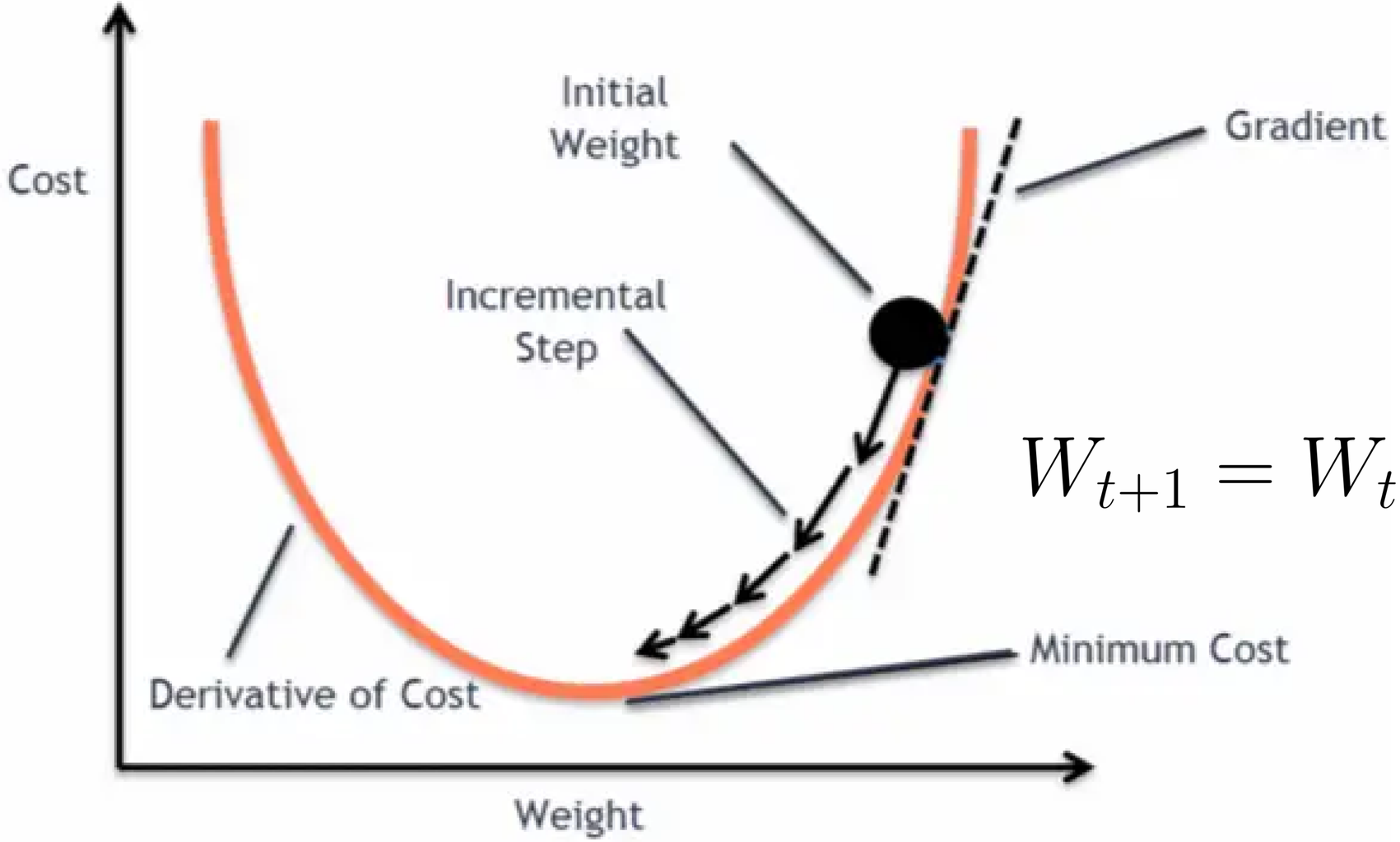
$$\nabla_W = \frac{\partial \mathcal{L}(W)}{\partial W}$$

update weights with this rule:

$$W_{t+1} = W_t - \alpha \nabla_{W_t}$$

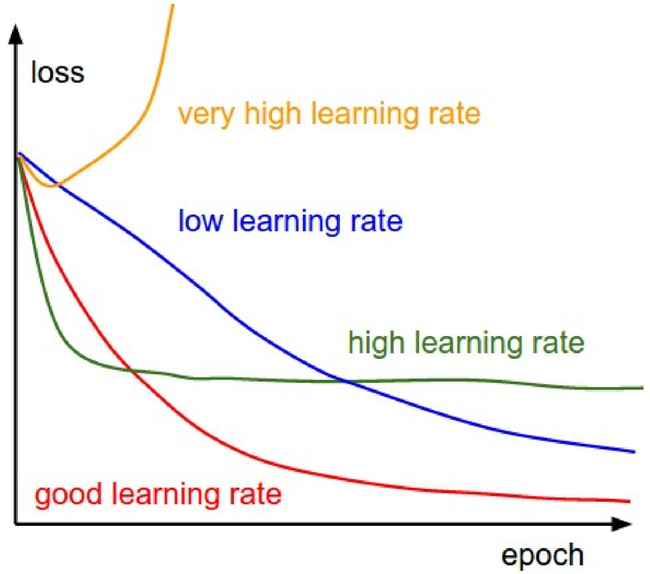


optimizers: sgd



$$W_{t+1} = W_t - \alpha \nabla_{W_t}$$

optimizers: learning rate



$$W_{t+1} = W_t - \alpha \nabla_{W_t}$$

$$\alpha = learning\ rate$$

Metrics

metrics are a way to evaluate performance

are used in the test of the model

some common metrics for classification are:

$$Accuracy \qquad \frac{T_p + T_n}{T_p + T_n + F_p + F_n}$$

$$Precision \qquad \frac{T_p}{T_p + F_p}$$

$$Recall \qquad \frac{T_p}{T_p + T_n}$$

Hands-on in colab

Colab link

discussion of results

Thank u <3