



DEEP LEARNING

MARCELO ERRECALDE (UNSL)

LAURA LANZARINI (UNLP)

CESAR ESTREBOU (UNLP)



MATERIALES Y ACTIVIDADES

- El material está disponible en IDEAS

<https://ideas.info.unlp.edu.ar/>

Contiene

- Diapositivas y links a la bibliografía.
- Actividades a realizar en clase.
- Autoevaluación del curso.



<https://goo.by/RwlClt>

BIBLIOGRAFÍA

- **Deep Learning with Python.**

François Chollet.

Manning, 2021

- **Neural Networks and Deep Learning**

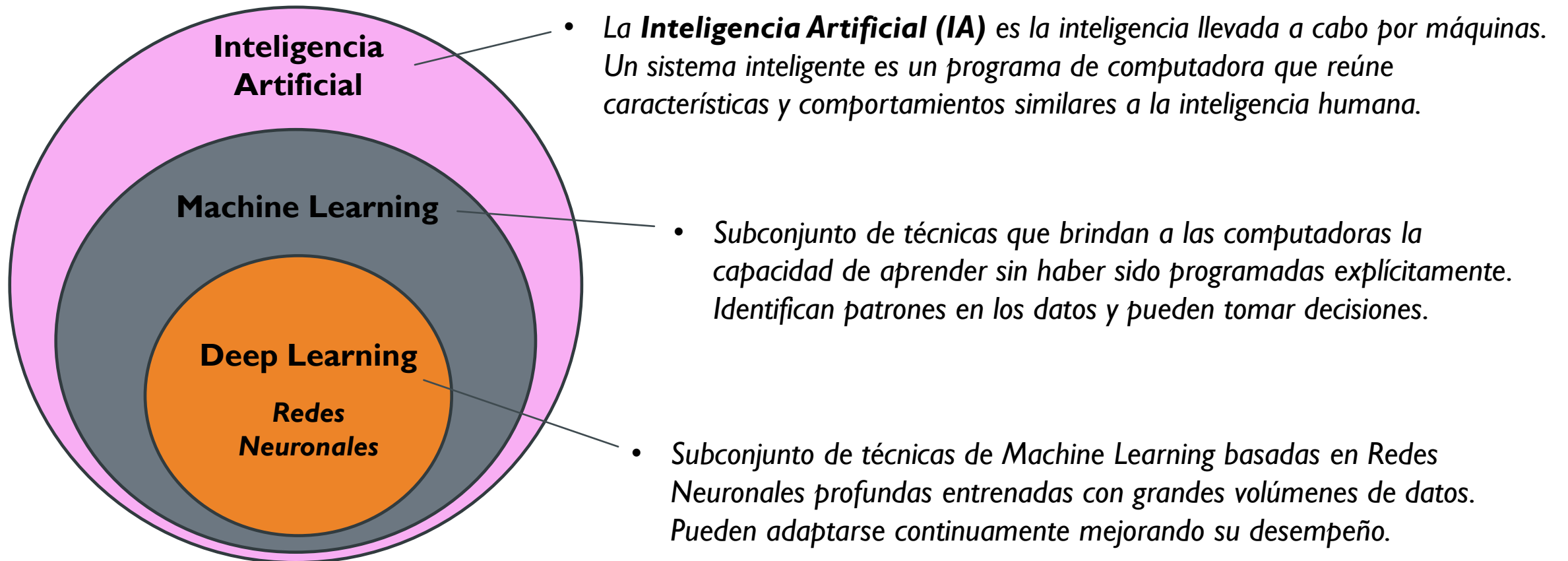
Michael A. Nielsen.

Determination Press, 2015

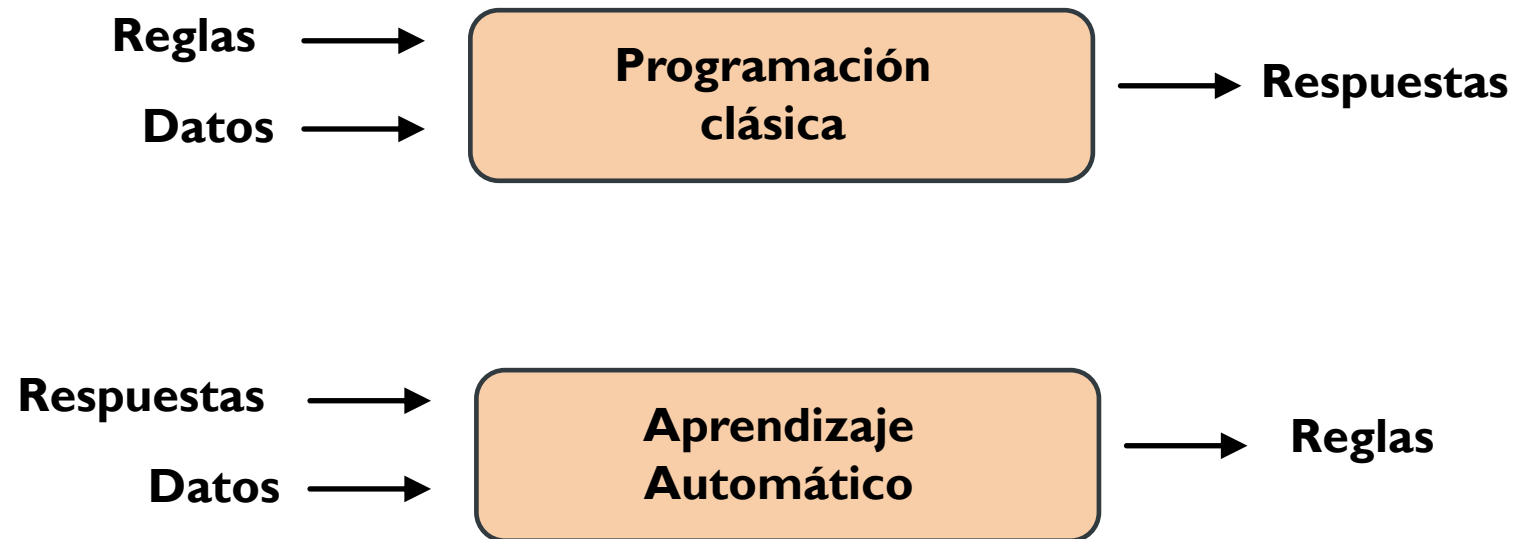


<https://goo.by/RwlClt>

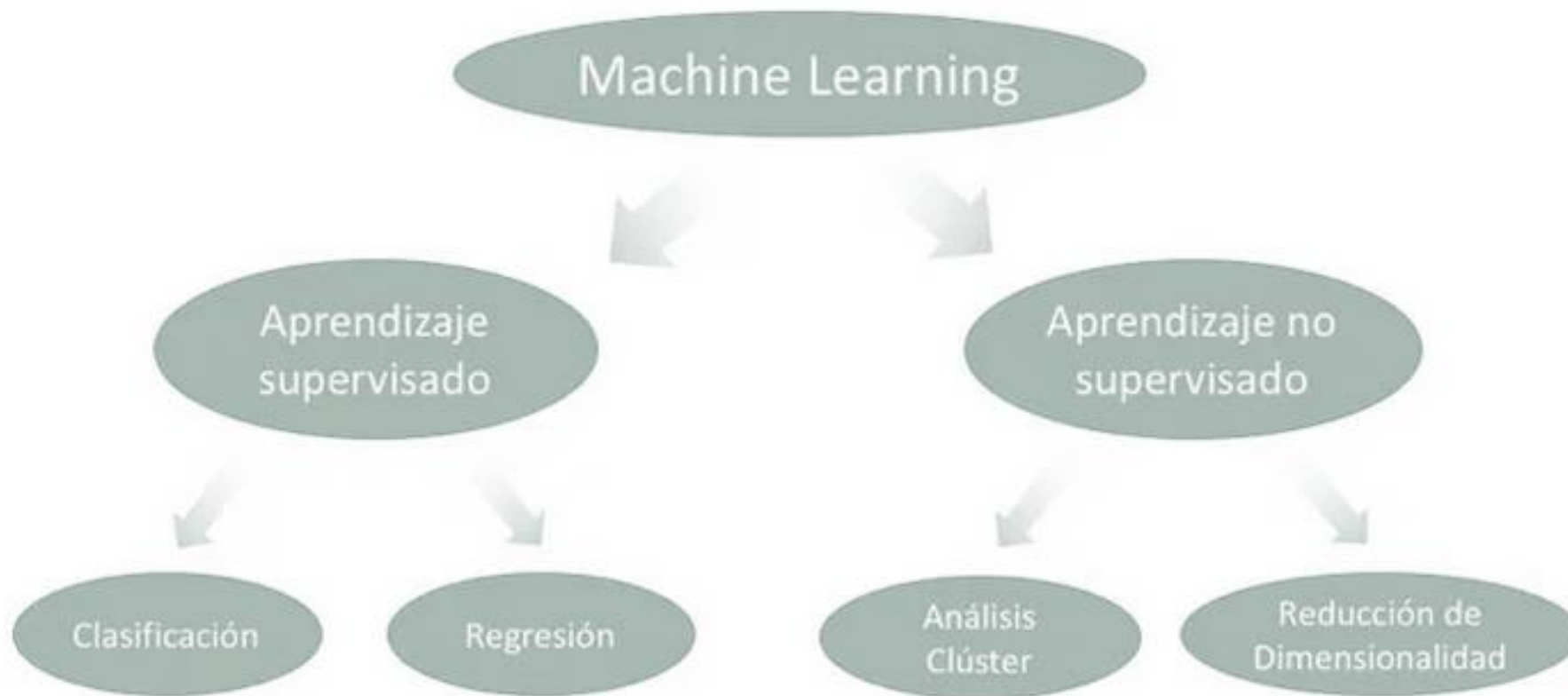
DEEP LEARNING



PROGRAMACIÓN CLÁSICA Y APRENDIZAJE AUTOMÁTICO



TIPOS DE APRENDIZAJE



APRENDIZAJE SUPERVISADO

GATO



GATO



GATO



ARBOL



ARBOL



CUADERNO



CUADERNO



CUADERNO



GATO



?

APRENDIZAJE NO SUPERVISADO



AGRUPAMIENTO

APRENDIZAJE SUPERVISADO

GATO



GATO



GATO



ARBOL



ARBOL



CUADERNO



CUADERNO



CUADERNO



GATO



?

Hoy trabajaremos con
APRENDIZAJE SUPERVISADO

RECONOCEDOR DE DÍGITOS ESCRITOS A MANO

- Cuando terminemos la clase de hoy habremos aprendido cómo entrenar una red neuronal para que pueda reconocer dígitos escritos a mano.

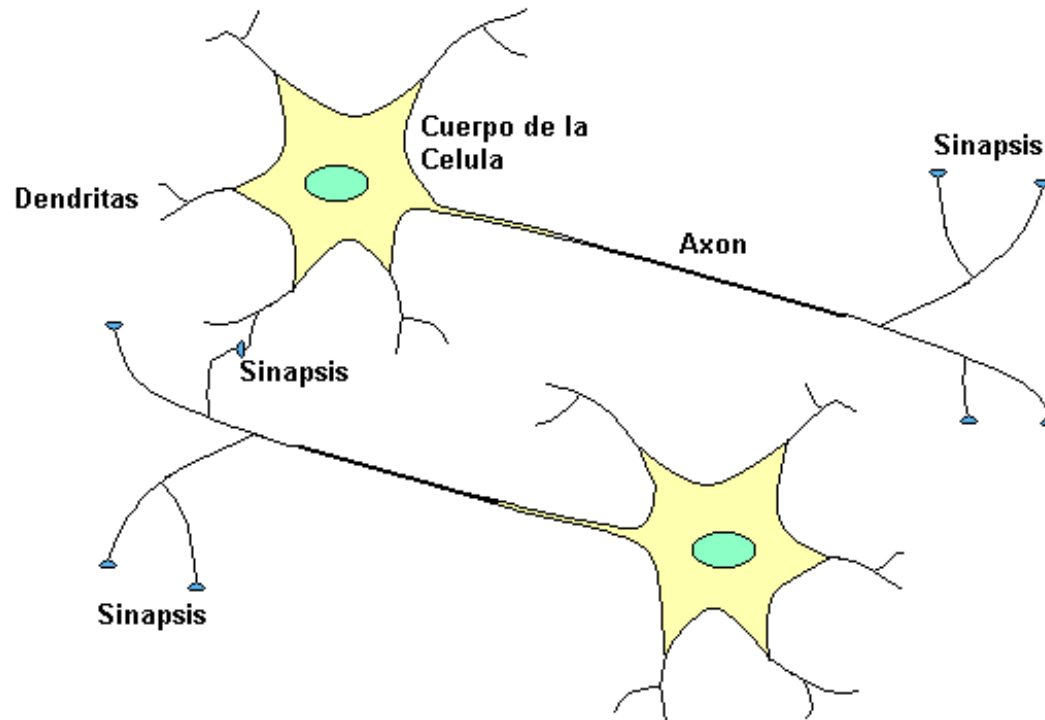
504192

REDES NEURONALES

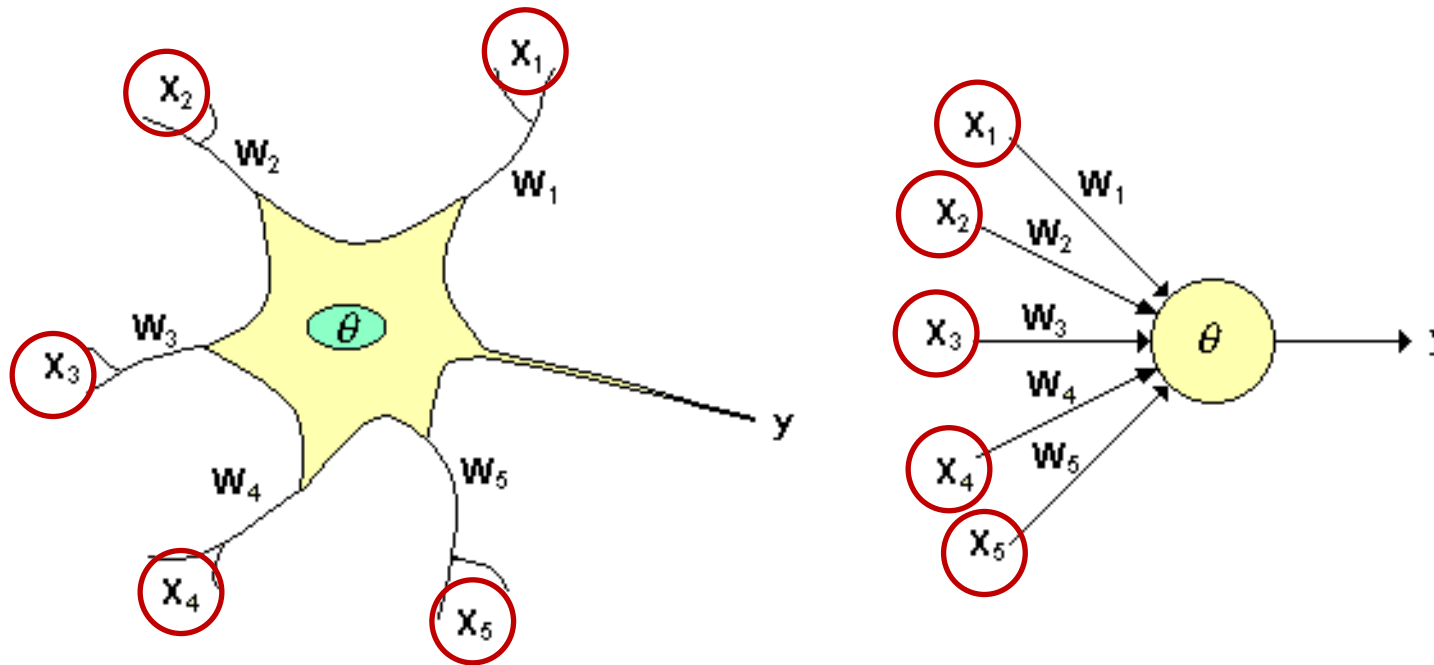
- El cerebro humano
 - Procesa información imprecisa rápidamente.
 - Aprende sin instrucciones explícitas.
 - Crea representaciones internas que permiten estas habilidades.
- Las Redes Neuronales Artificiales o simplemente **Redes Neuronales**, buscan emular el comportamiento del cerebro humano.

NEURONA BIOLÓGICA

- El cerebro consta de un gran número de elementos (aprox. 10^{11}) altamente interconectados (aprox. 10^4 conexiones por elemento), llamados **neuronas**.

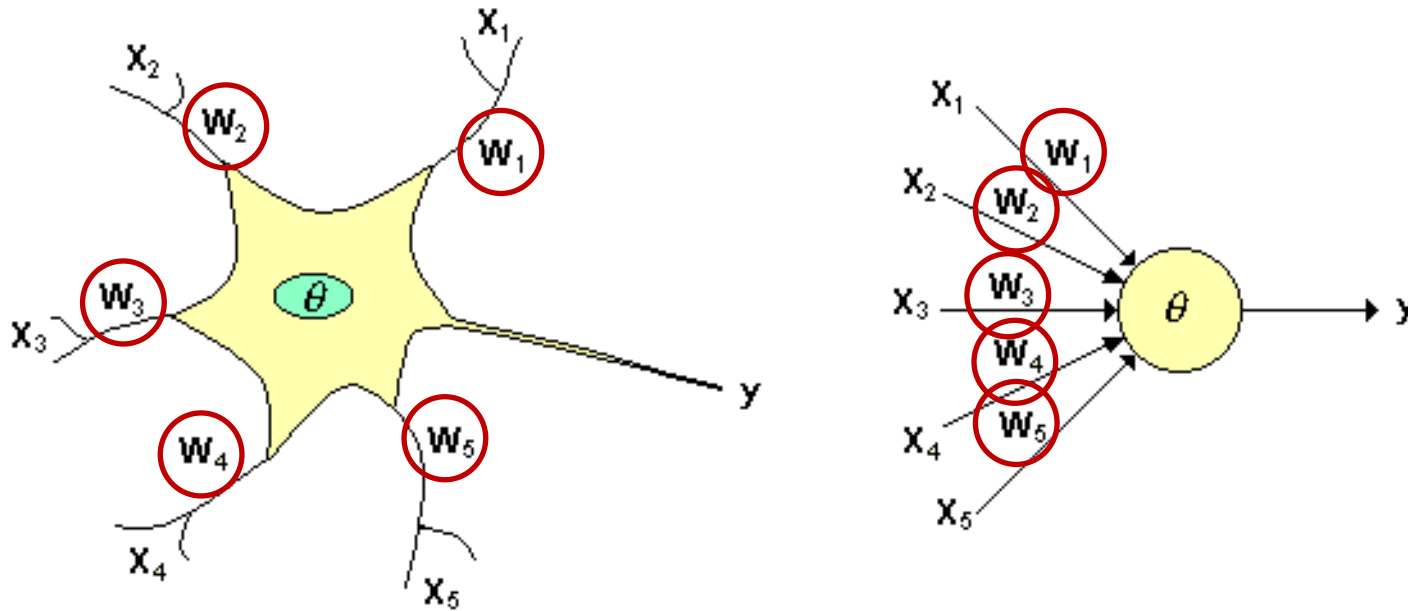


SIMILITUDES ENTRE UNA NEURONA BIOLÓGICA Y UNA ARTIFICIAL



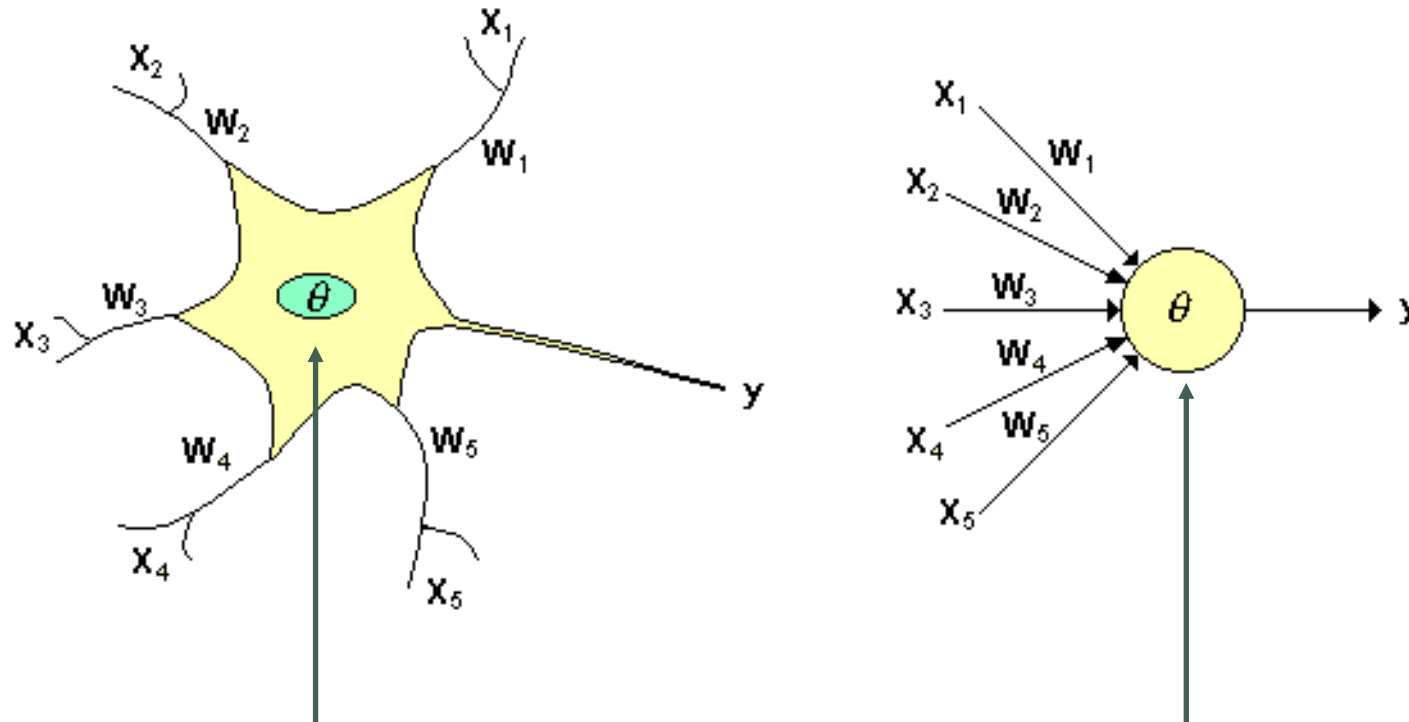
Las entradas x_i representan las señales que provienen de otras neuronas y que son capturadas por las dendritas

SIMILITUDES ENTRE UNA NEURONA BIOLÓGICA Y UNA ARTIFICIAL



Los pesos w_i son la intensidad de la sinapsis que conecta dos neuronas

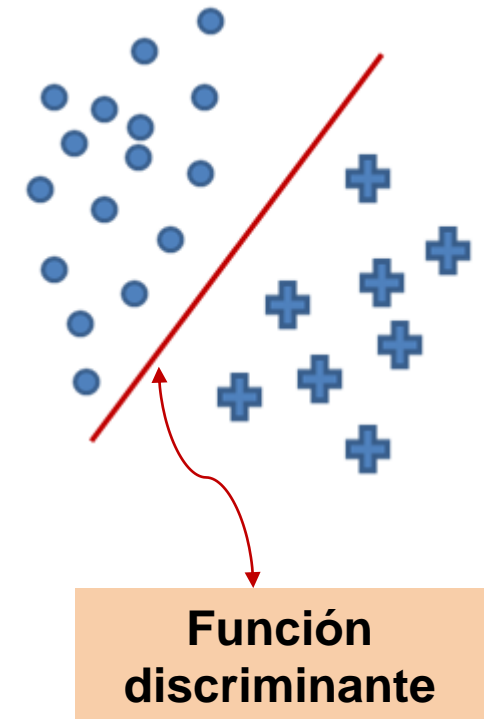
SIMILITUDES ENTRE UNA NEURONA BIOLÓGICA Y UNA ARTIFICIAL



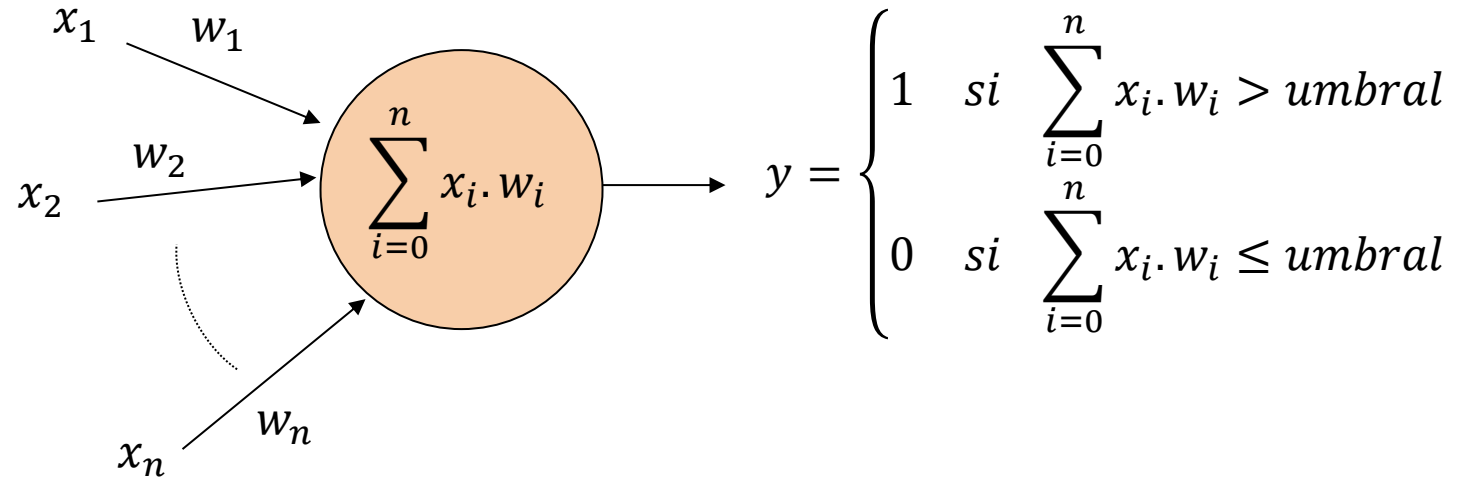
θ es la función umbral que la neurona debe sobrepasar para activarse; este proceso ocurre biológicamente en el cuerpo de la célula

EL PERCEPTRÓN

- Es una RN formada por una única neurona.
- Utiliza aprendizaje supervisado.
- Su regla de aprendizaje es una modificación de la propuesta por Hebb.
- Se adapta teniendo en cuenta el error entre la salida que da la red y la salida esperada.
- Representa una única función discriminante que separa **linealmente** los ejemplos en **dos** clases.



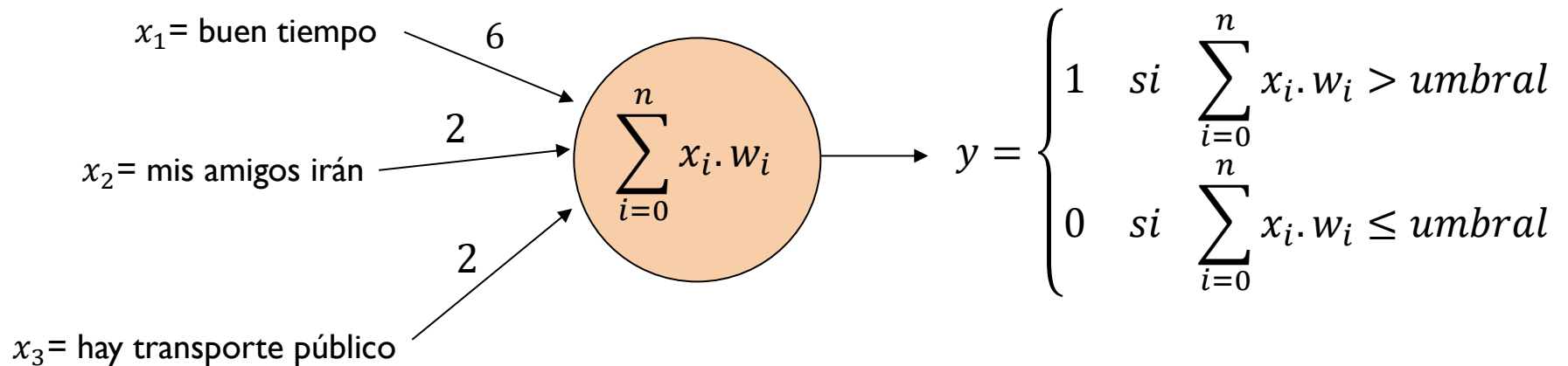
FUNCIONAMIENTO DE UN PERCEPTRÓN



EJEMPLO: Perceptrón para decidir si debo ir o no a un evento

- Cada entrada x_i vale 1 si se cumple la condición y 0 si no.
- Si la salida y es 1 significa que debo ir al evento.

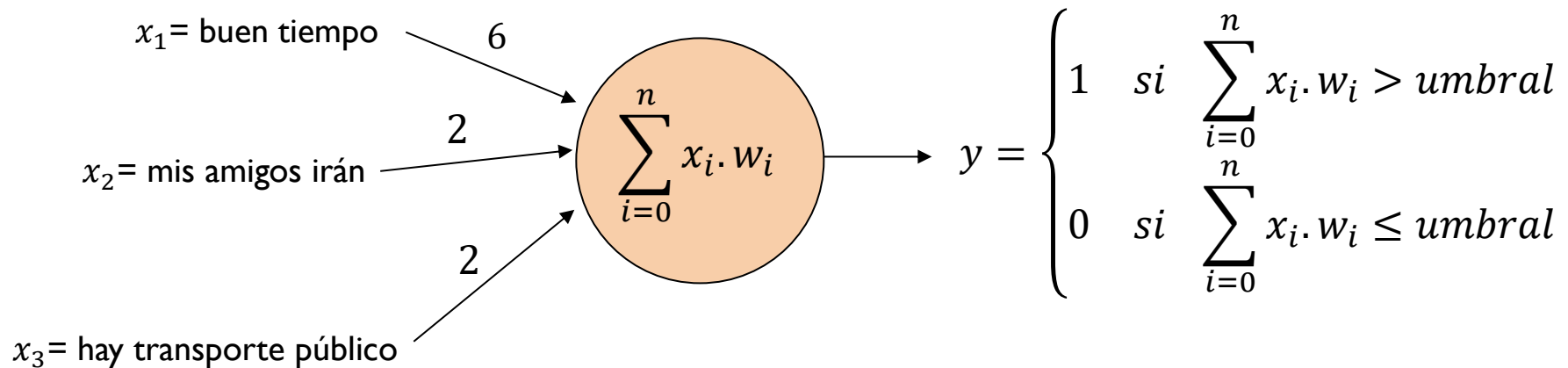
$$Umbral = 3$$



EJEMPLO: Perceptrón para decidir si debo ir o no a un evento

- Cada entrada x_i vale 1 si se cumple la condición y 0 si no.
- Si la salida y es 1 significa que debo ir al evento.

$$Umbral = 3$$

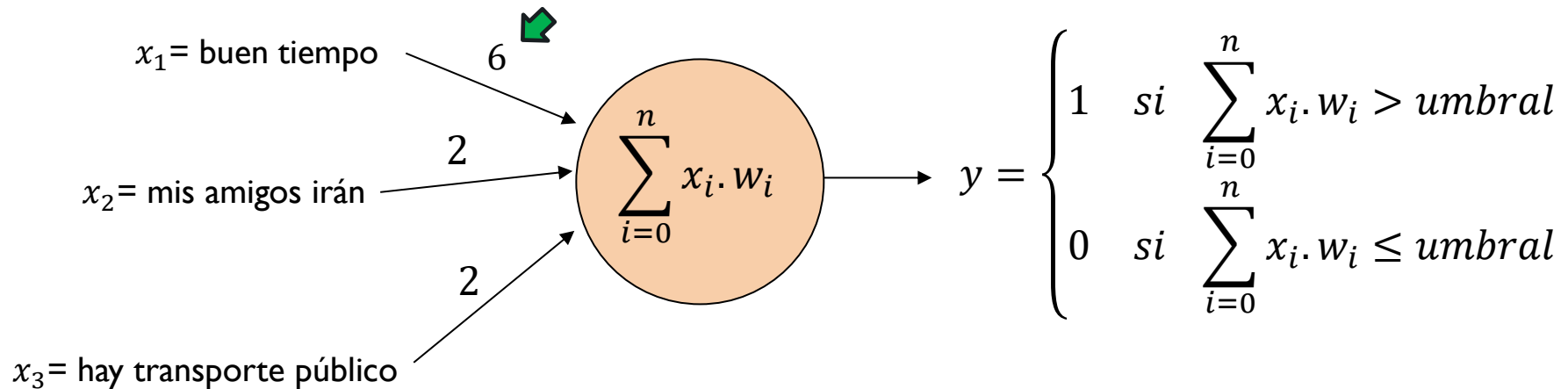


**¿Qué pasa si cambiamos
 $Umbral = 5$?**

EJEMPLO: Perceptrón para decidir si debo ir o no a un evento

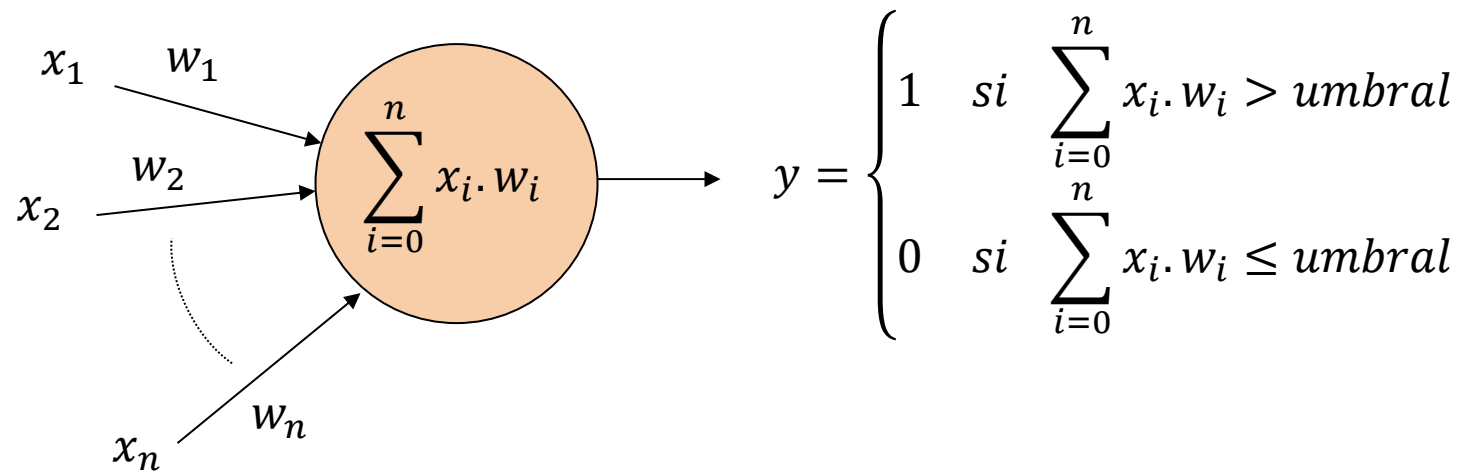
- Cada entrada x_i vale 1 si se cumple la condición y 0 si no.
- Si la salida y es 1 significa que debo ir al evento.

$$Umbral = 3$$

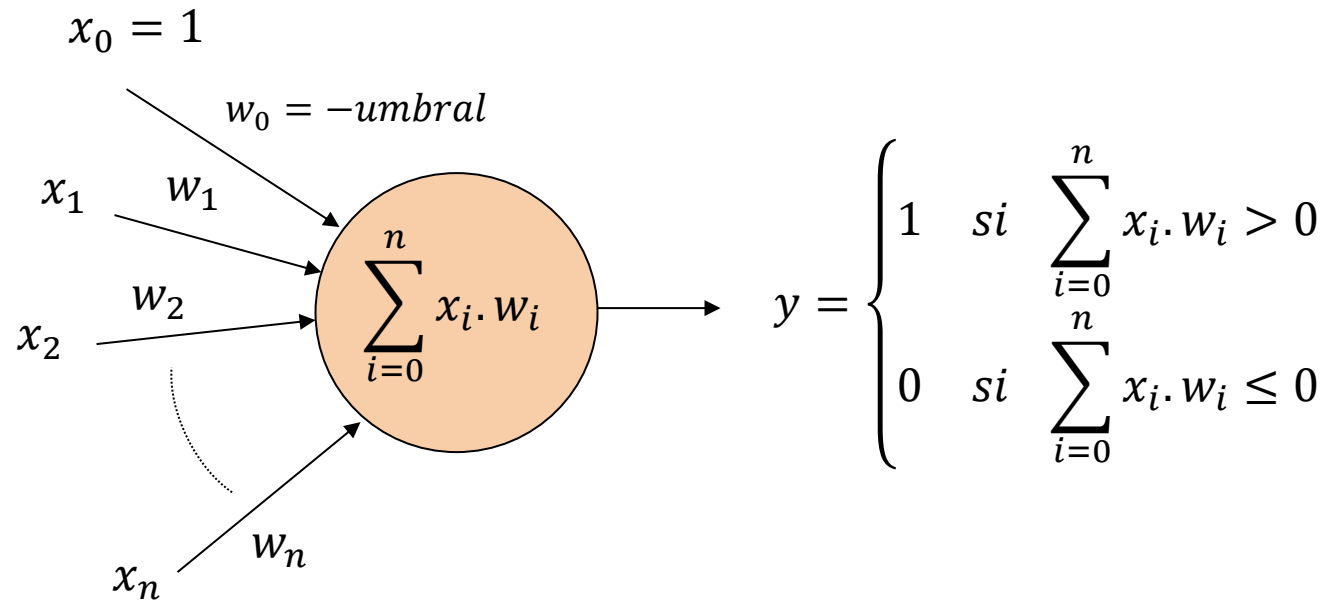


¿Qué pasa si $W_1=2$?

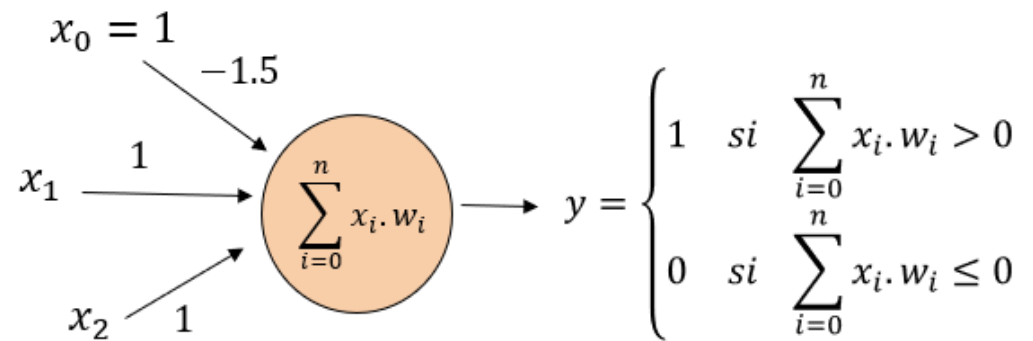
EL PERCEPTRÓN



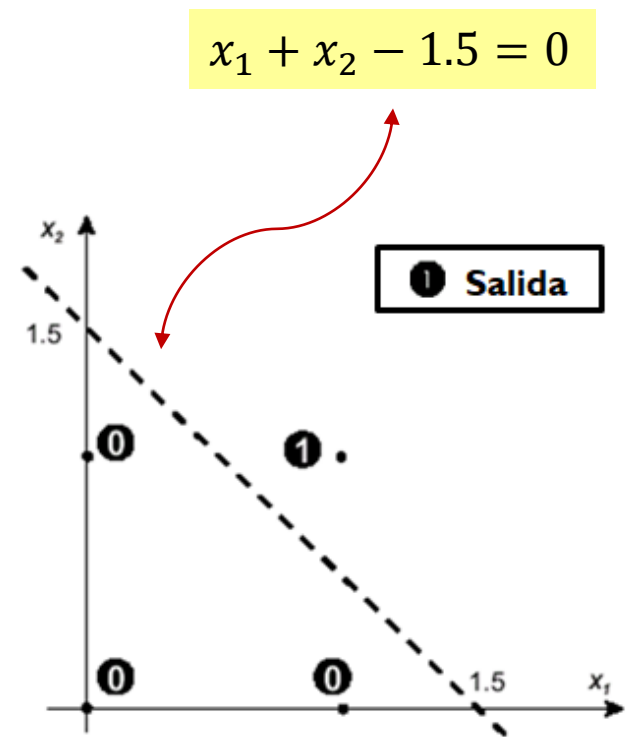
EL PERCEPTRÓN



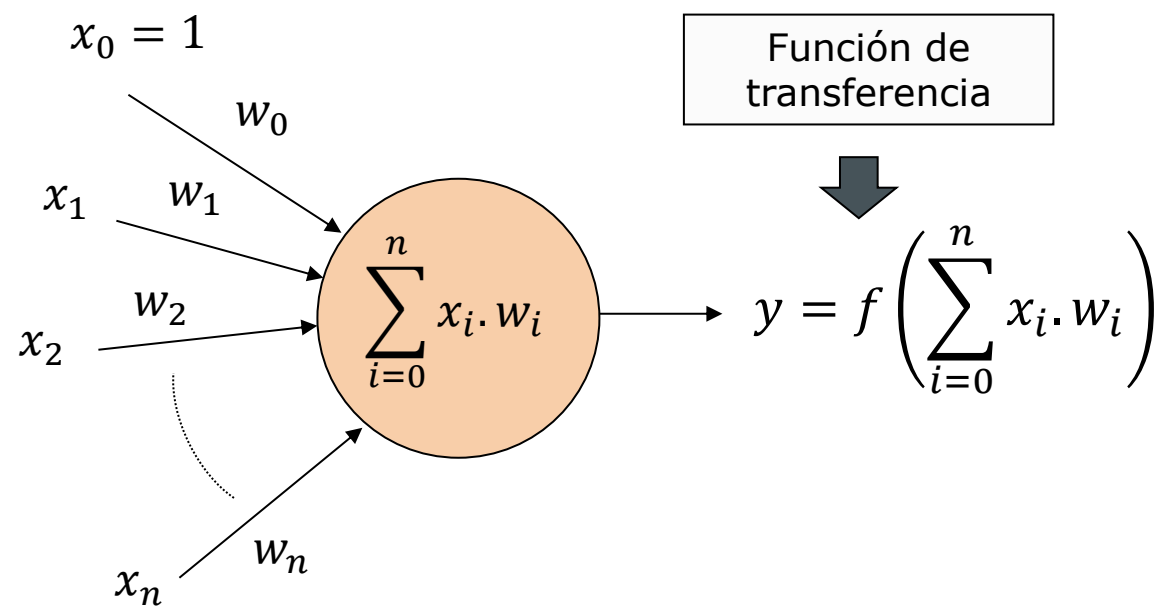
AND



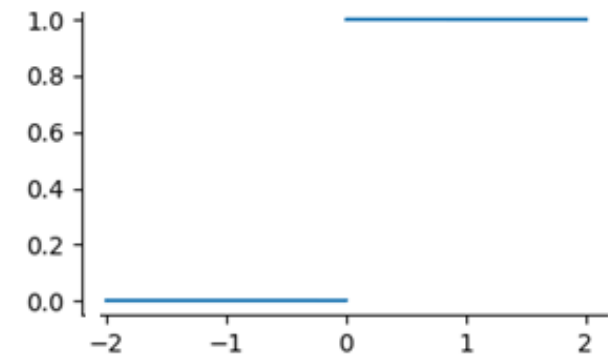
x_1	x_2	Σ	y
0	0	-1.5	0
0	1	-0.5	0
1	0	-0.5	0
1	1	0.5	1



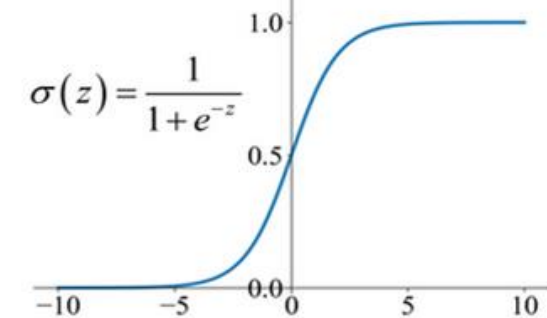
NEURONA ARTIFICIAL



umbral



Sigmoid

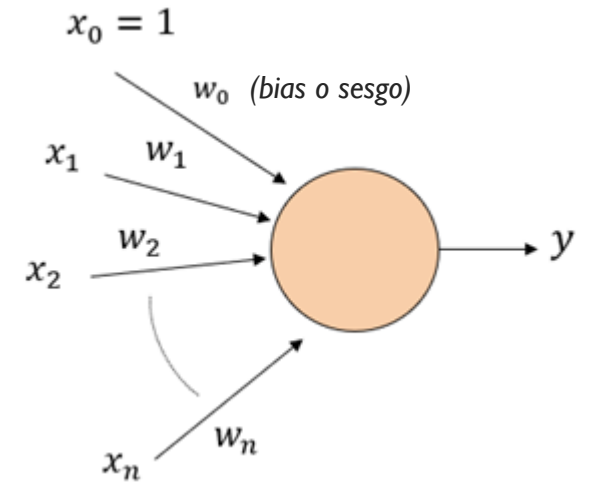


ENTRENAMIENTO DE UNA NEURONA

- Indicar el valor de la tasa de aprendizaje (*learning_rate*)
- Inicializar los pesos W y el sesgo con valores random
- Mientras (la calidad de la respuesta no sea aceptable)
 - Para cada ejemplo
 - Ingresar el ejemplo a la red y obtener la salida y
 - Calcular el error cometido y determinar los cambios a realizar.
 - Actualizar los pesos de la neurona de manera adecuada.

Función de Costo

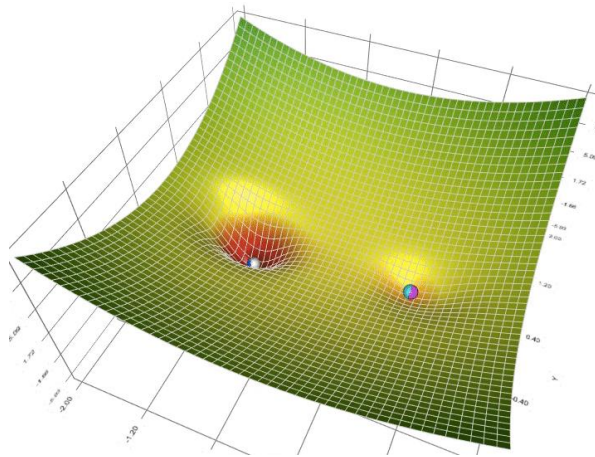
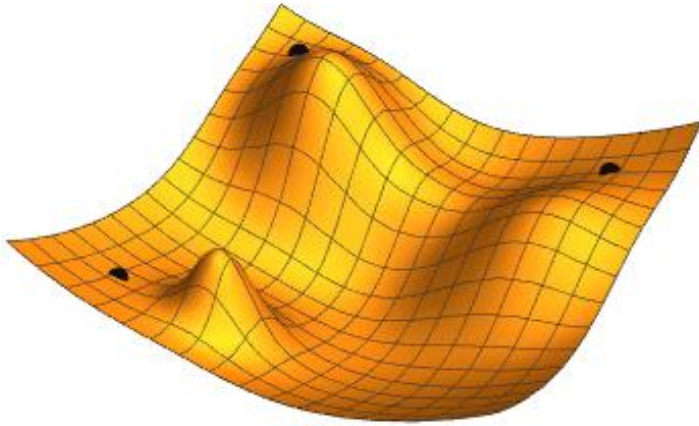
Optimización por gradiente
(se buscará el mínimo de la función de costo)



FUNCIÓN DE COSTO

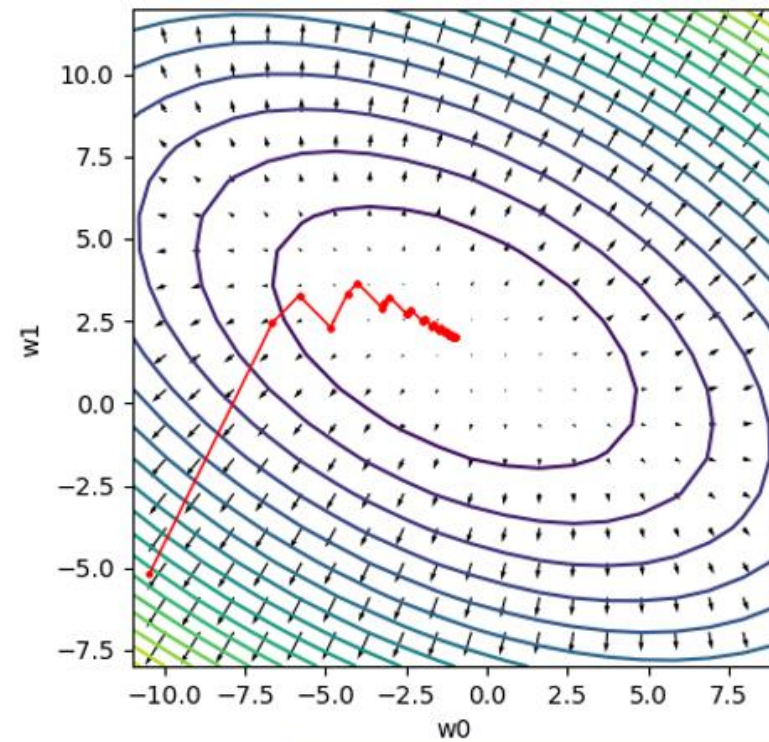
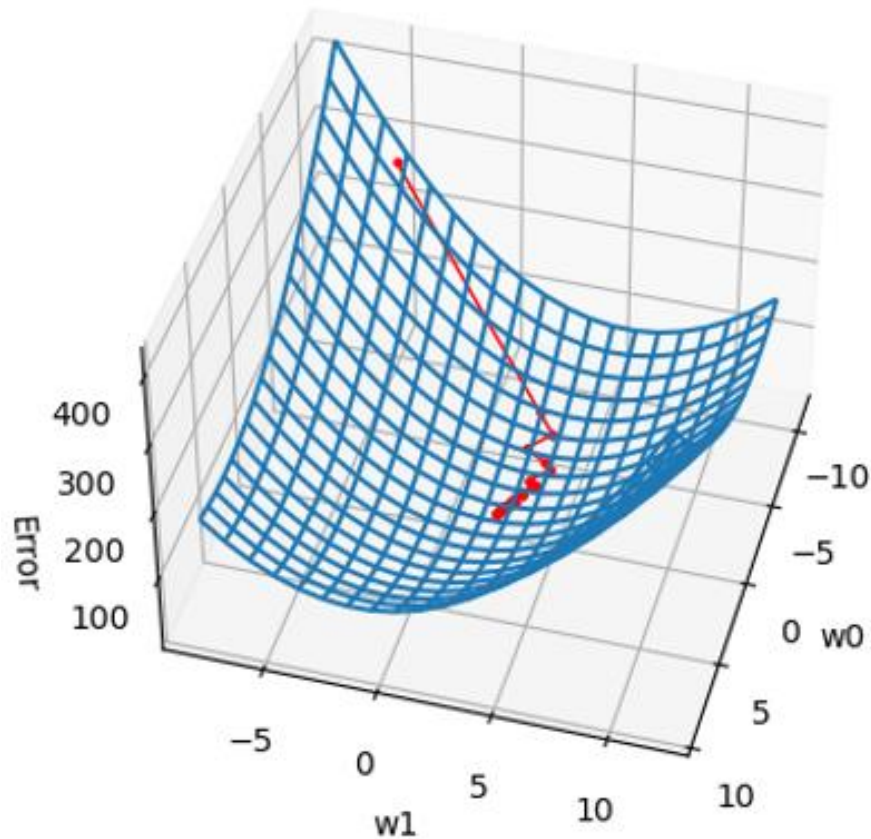
- Es una medida de la diferencia entre las predicciones del modelo y los valores reales.
- Indica qué tan bien está funcionando el modelo.
- Funciones de costo para problemas de clasificación
 - **Entropía cruzada binaria** (para problemas de clasificación binaria, 0 o 1)
 - **Entropía cruzada categórica** (para problemas multiclase)

OPTIMIZACIÓN POR GRADIENTE



- Es un estrategia eficiente y fundamentada matemáticamente para encontrar los valores de las variables que minimizan (o maximizan) una función
- **Dirección de mayor cambio:** El gradiente de una función en un punto específico es un vector que indica la dirección del mayor incremento de la función.
- **Eficiencia:** El gradiente proporciona información local sobre la pendiente de la función. Computacionalmente eficiente para problemas complejos o con muchas variables.
- **Convergencia** a un óptimo local.
- **Adaptabilidad:** Existen variaciones del gradiente descendente (como el gradiente estocástico, mini-lote o con momentos) que lo hacen más robusto y aplicable a distintos tipos de problemas, incluyendo funciones no convexas o grandes volúmenes de datos.

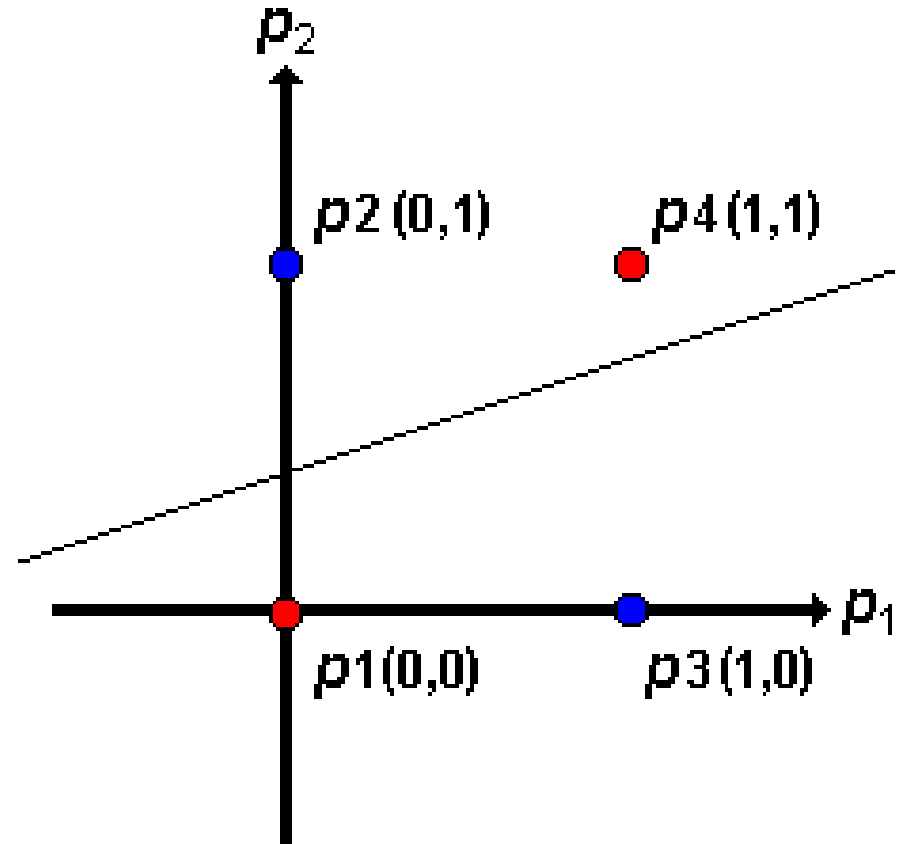
MINIMIZACIÓN DE LA FUNCIÓN DE ERROR USANDO DESCENSO DE GRADIENTE ESTOCÁSTICO



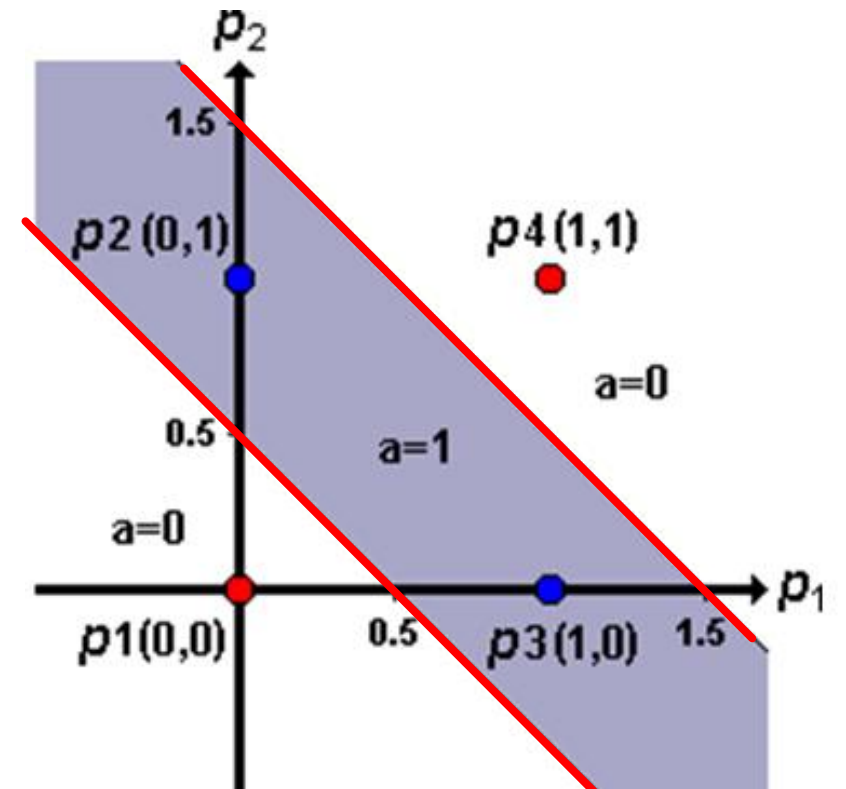
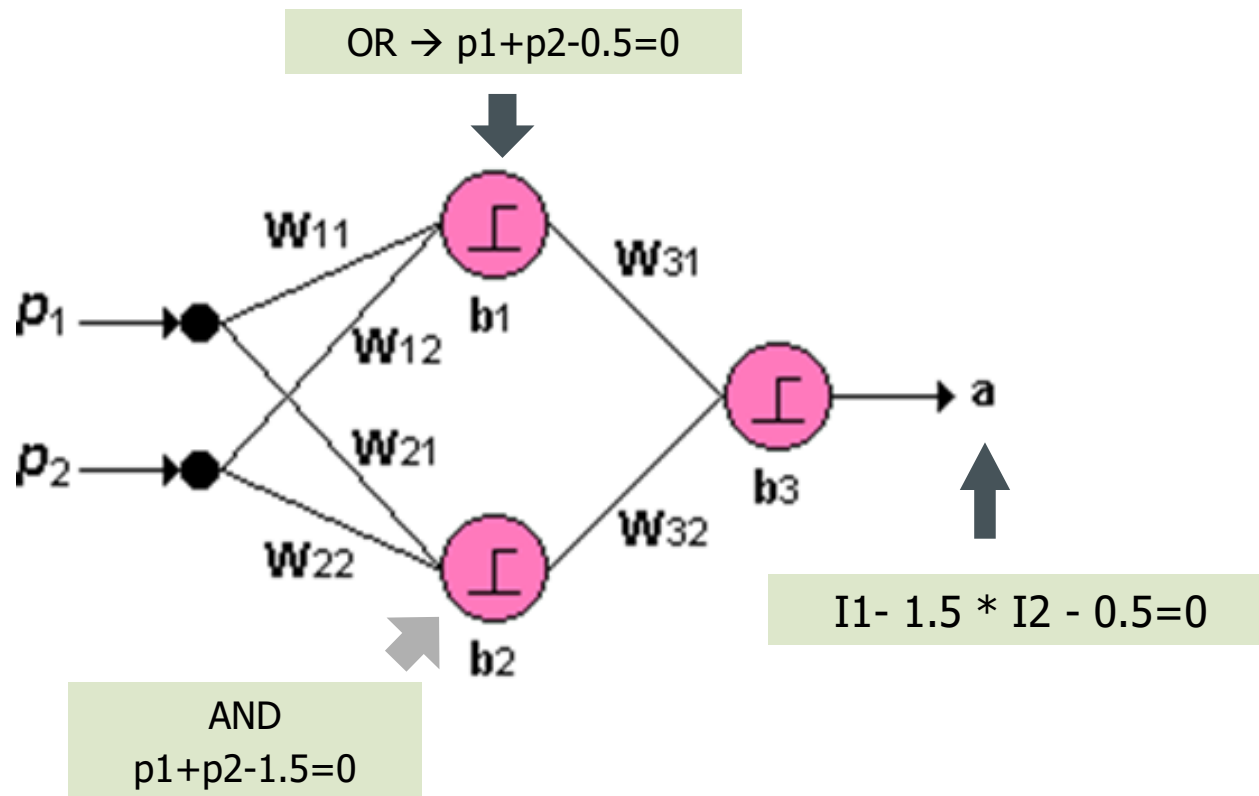
`gradienteFuncion4_CombinadorLineal.py`

REDES MULTICAPA

- Con una neurona puede resolverse un problema de dos clases linealmente separables.
- Hay problemas de 2 clases que no cumplen esta condición.
- El problema del **XOR**, si bien tiene dos clases, no es linealmente separable. Por lo tanto, no alcanza con una única neurona para resolverlo.

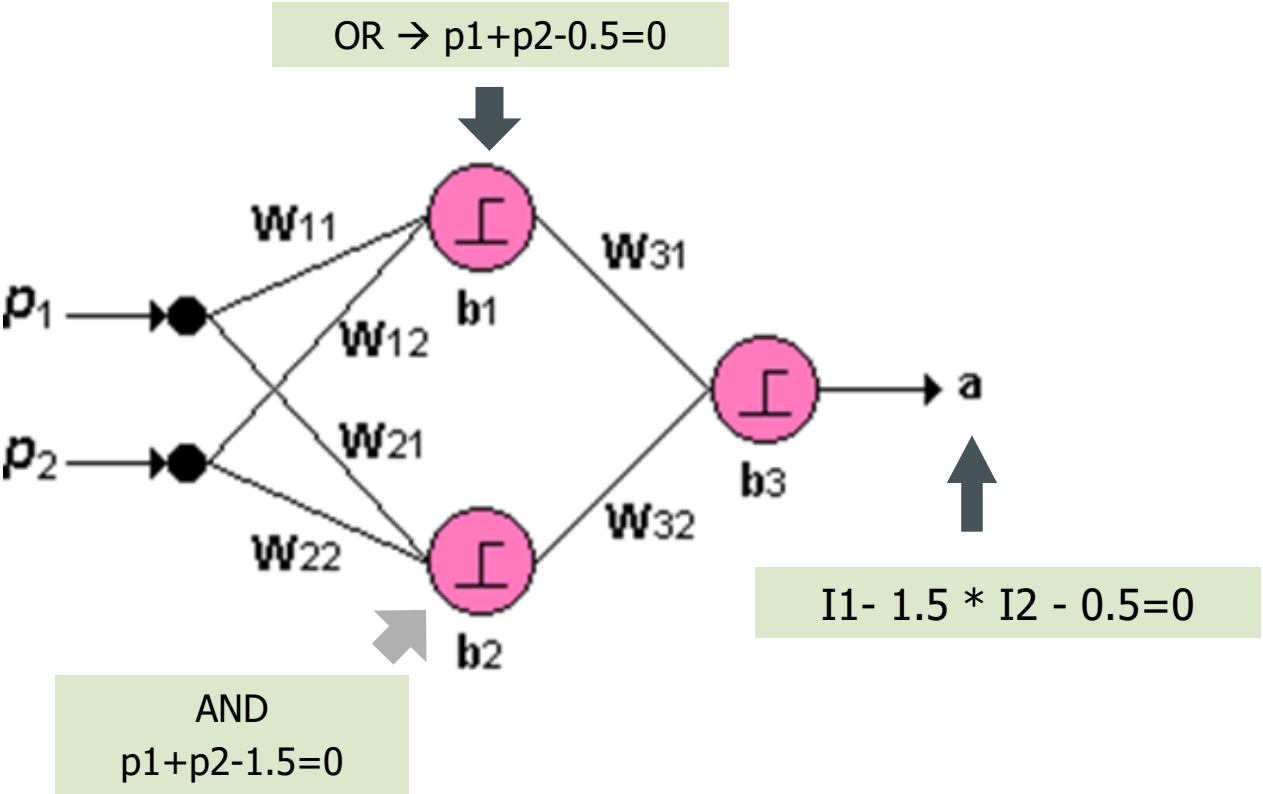


XOR

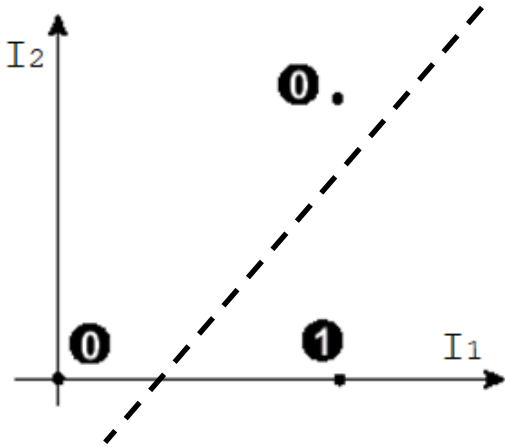


$$w_{11}=1 \quad w_{12}=1 \quad b_1=-0.5 \quad ; \quad w_{21}=1 \quad w_{22}=1 \quad b_2=-1.5 \quad ; \quad w_{31}=1 \quad w_{32}=-1.5 \quad b_3=-0.5$$

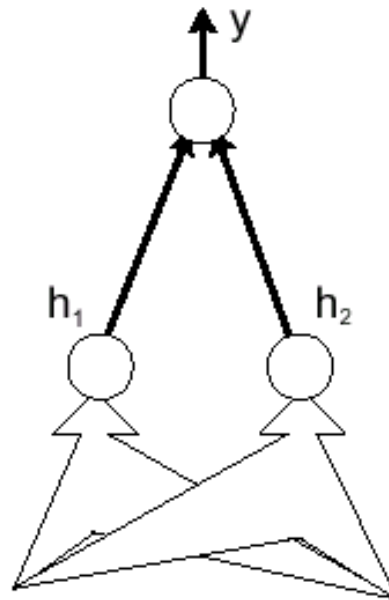
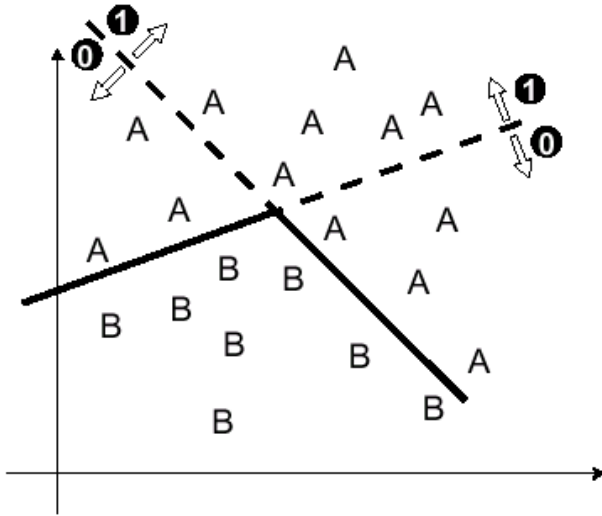
XOR



p_1	p_2	I_1 (or)	I_2 (AND)	a
1	0	1	0	1
1	1	1	1	0
0	0	0	0	0
0	1	1	0	1

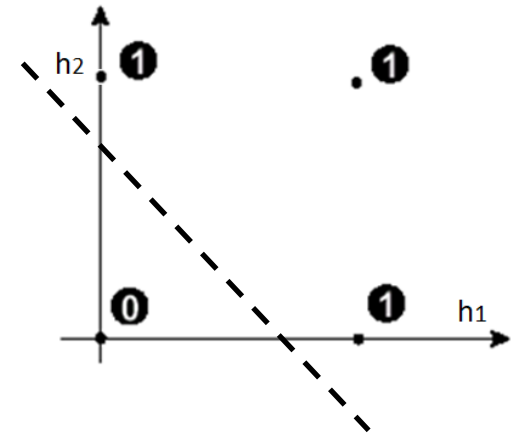


PROBLEMA NO SEPARABLE LINEALMENTE



h_1	h_2	y
0	0	0
0	1	1
1	0	1
1	1	1

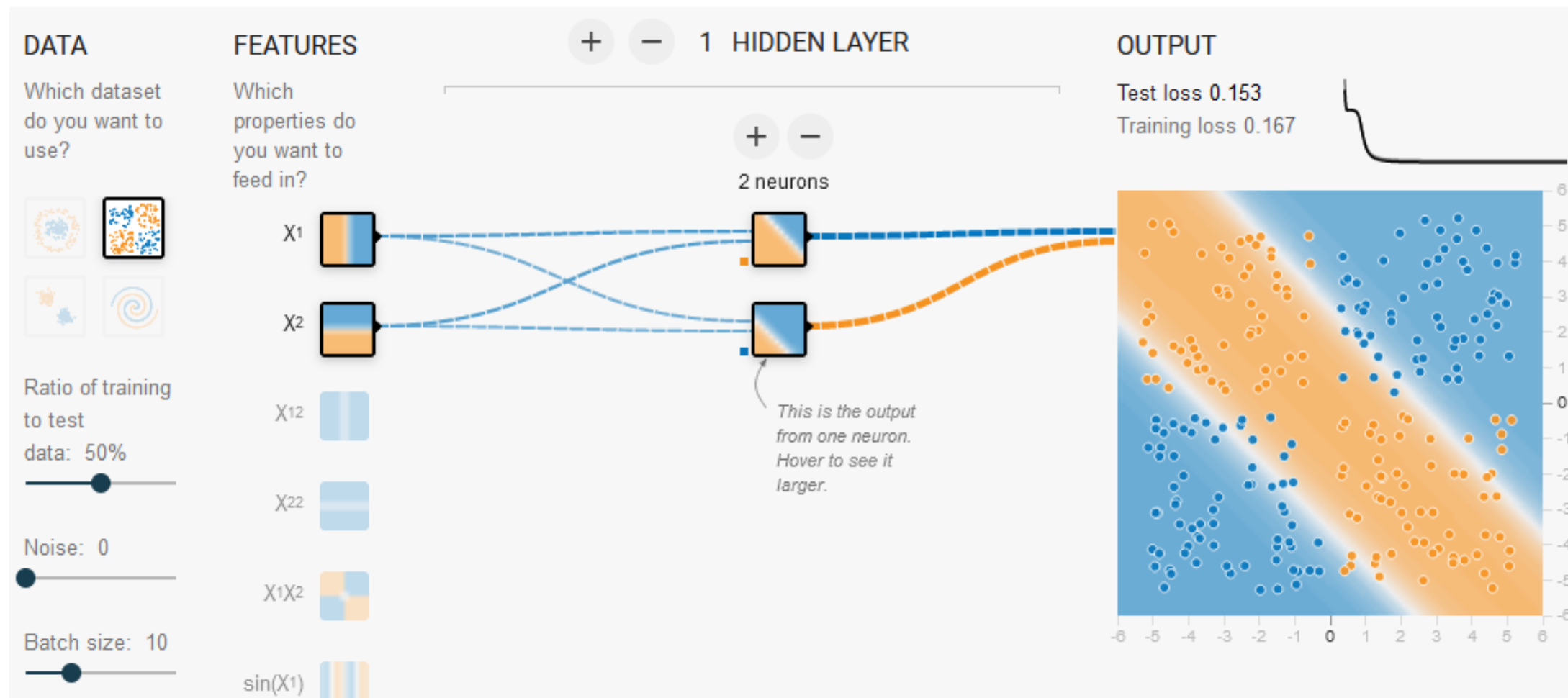
$\left. \vphantom{\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{matrix}} \right\} A \leftrightarrow 1$



- Se utiliza un algoritmo general que integra el aprendizaje entre las dos capas.

ANIMACIÓN DE UNA RN

Tinker With a Neural Network Right Here in Your Browser



RESOLUCIÓN DE UNA TAREA DE CLASIFICACIÓN

- Conjunto de datos etiquetados (aprendizaje supervisado) ←
- Definición de la arquitectura de la red
 - Número de capas y tamaño de cada una
 - Función de activación a usar en cada capa
- Entrenamiento
 - Función de costo
 - Técnica de optimización para reducir el error
- Evaluar el modelo

EJEMPLO: CLASIFICACIÓN DE FLORES DE IRIS



<https://archive.ics.uci.edu/ml/datasets/Iris>

DATASET IRIS

Id	sepalength	sepalwidth	petallength	petalwidth	class
1	5,1	3,5	1,4	0,2	Iris-setosa
2	4,9	3,0	1,4	0,2	Iris-setosa
...
95	5,6	2,7	4,2	1,3	Iris-versicolor
96	5,7	3,0	4,2	1,2	Iris-versicolor
97	5,7	2,9	4,2	1,3	Iris-versicolor
...
149	6,2	3,4	5,4	2,3	Iris-virginica
150	5,9	3,0	5,1	1,8	Iris-virginica

<https://archive.ics.uci.edu/ml/datasets/Iris>

NORMALIZACION DE ATRIBUTOS

La normalización permite expresar los valores de los atributos sin utilizar las unidades de medida originales facilitando su comparación y uso conjunto

Normalización lineal uniforme

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- Es muy sensible a valores atípicos.
- Los valores quedan expresados en el intervalo $[0, 1]$

Normalización usando media y desvío

$$X' = \frac{X - media(X)}{desviacion(X)}$$

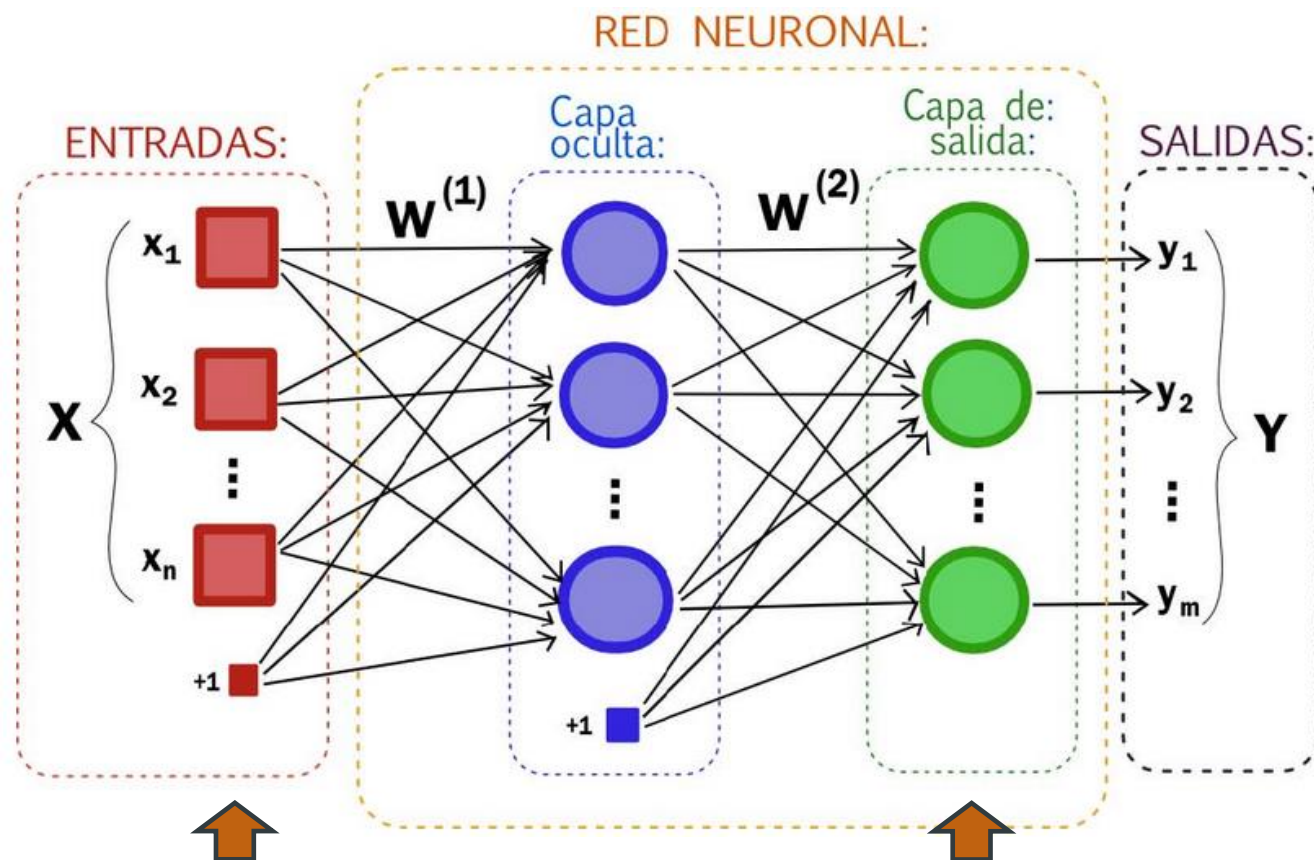
- Los valores quedan distribuidos normalmente alrededor de 0 con desviación 1

RESOLUCIÓN DE UNA TAREA DE CLASIFICACIÓN

- Conjunto de datos etiquetados (aprendizaje supervisado)
- Definición de la arquitectura de la red
 - Número de capas y tamaño de cada una
 - Función de activación a usar en cada capa
- Entrenamiento
 - Función de costo
 - Técnica de optimización para reducir el error
- Evaluar el modelo

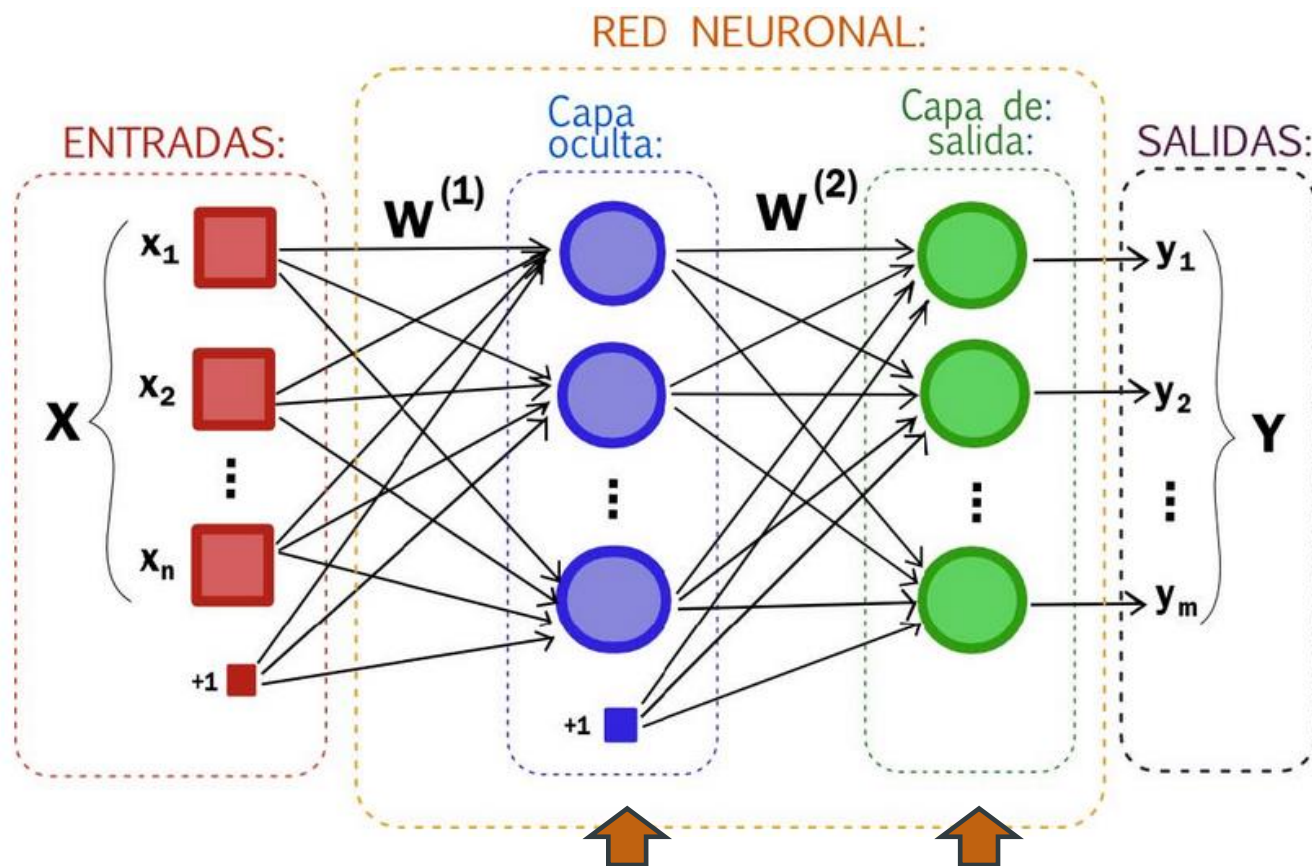


ARQUITECTURA DE LA RED



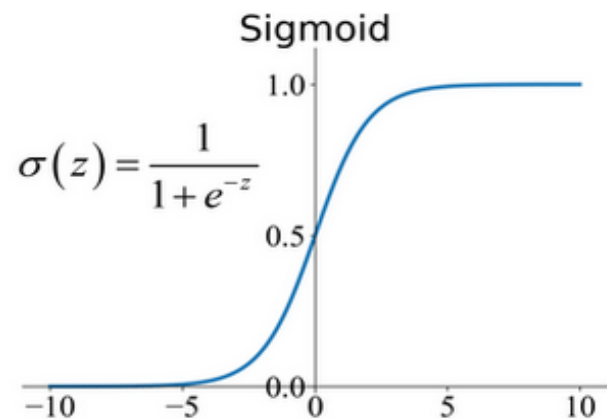
Las dimensiones de las capas de entrada y salida las define el problema

ARQUITECTURA DE LA RED

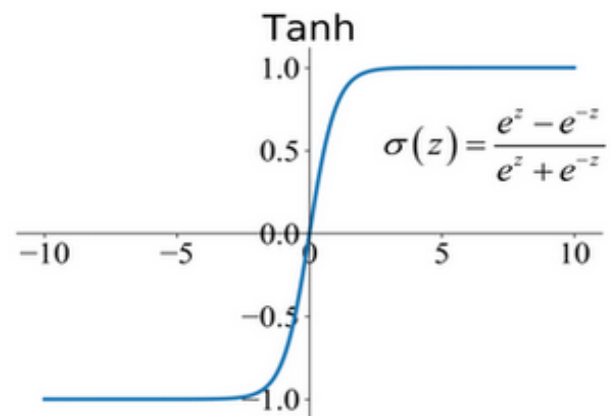
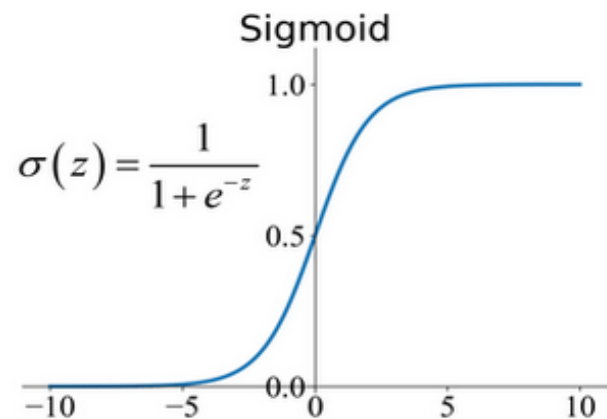


Su respuesta depende de la **Función de activación** elegida

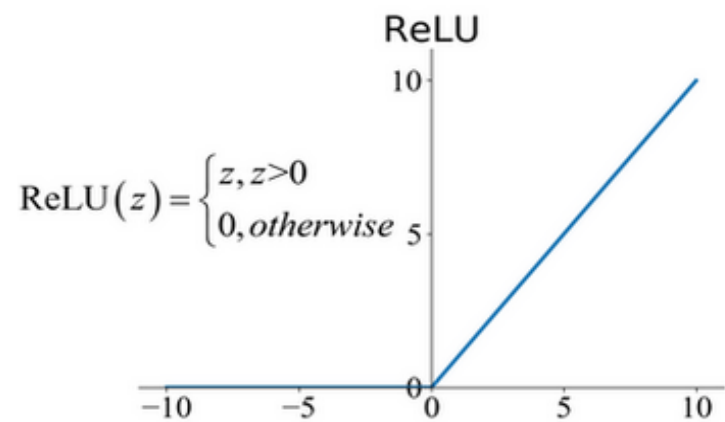
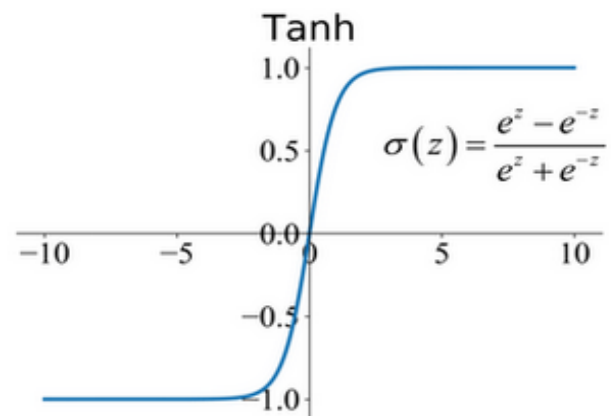
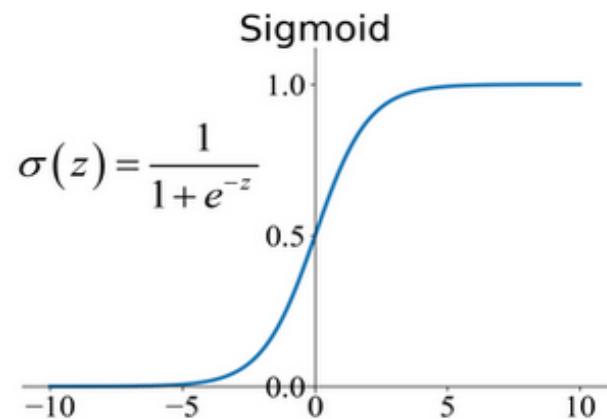
FUNCIONES DE ACTIVACIÓN



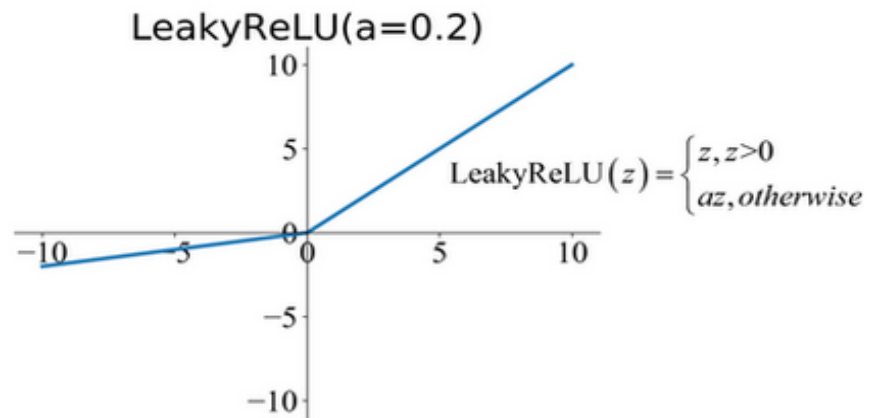
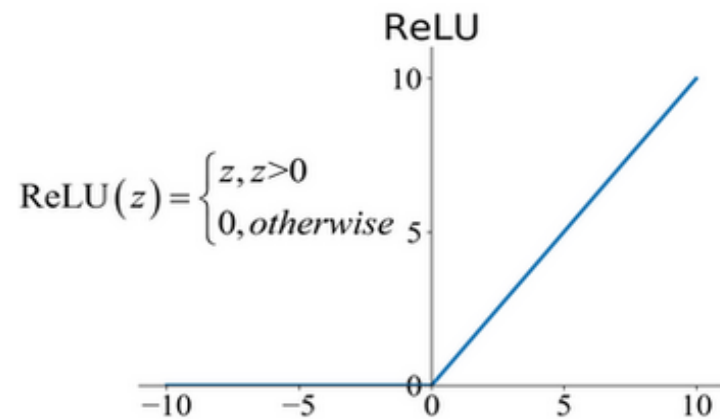
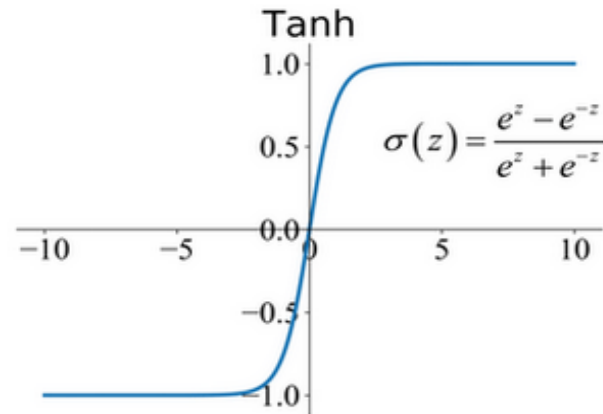
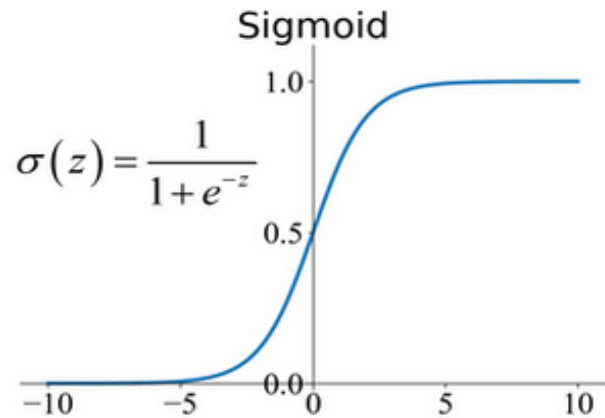
FUNCIONES DE ACTIVACIÓN



FUNCIONES DE ACTIVACIÓN

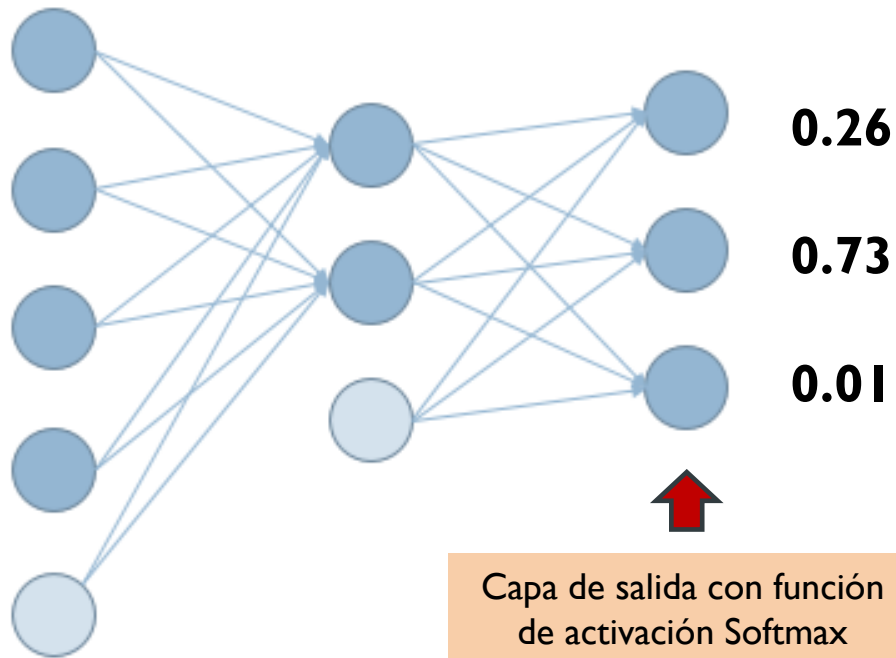


FUNCIONES DE ACTIVACIÓN



FUNCIÓN SOFTMAX

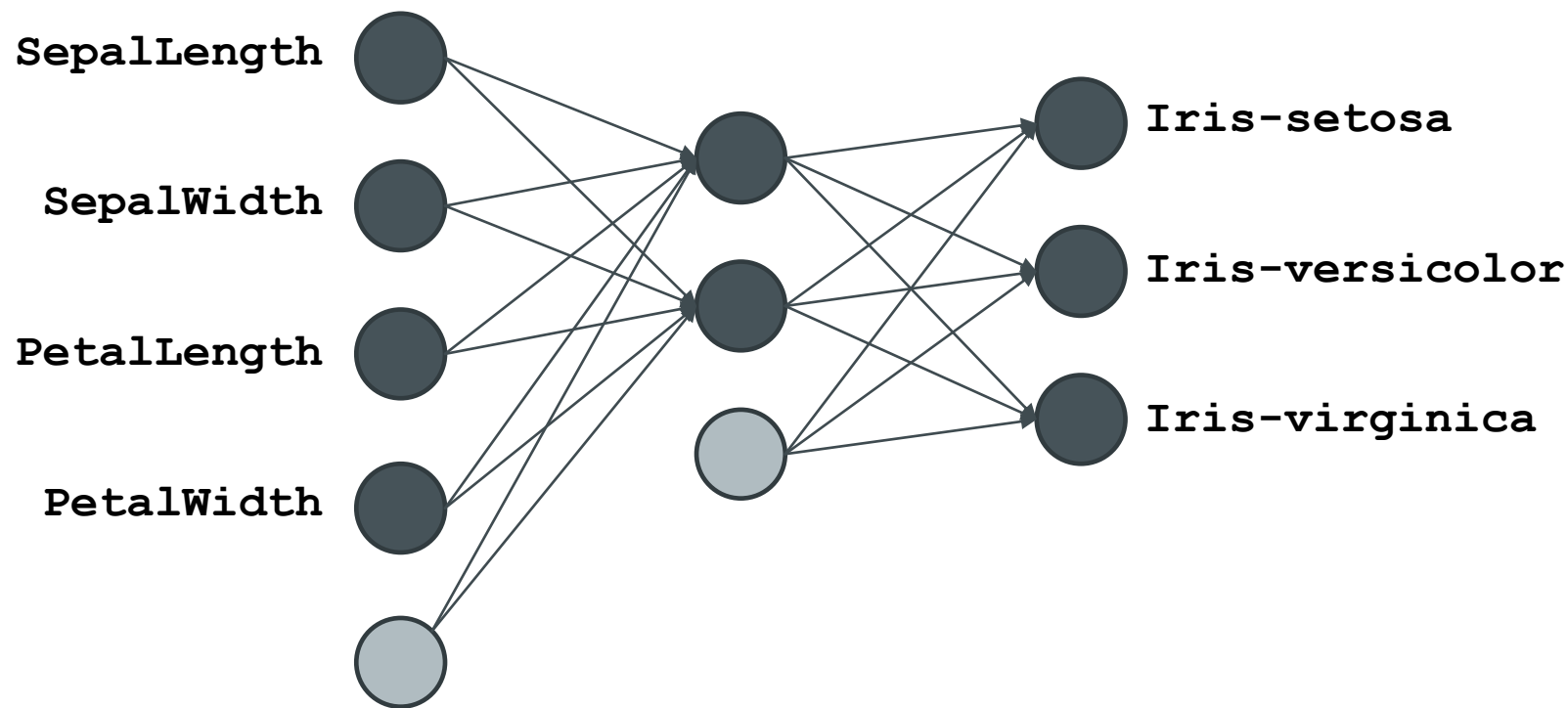
- La función softmax convierte el vector de salidas de una red neuronal en probabilidades.
- Se utiliza generalmente en la **última capa** de modelos de clasificación **multiclase**.



$$\text{Softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^n e^{z_k}}$$

- z_j es la entrada de la j -ésima neurona
- n es el número total de clases

ARQUITECTURA A UTILIZAR PARA CLASIFICAR LAS FLORES DE IRIS

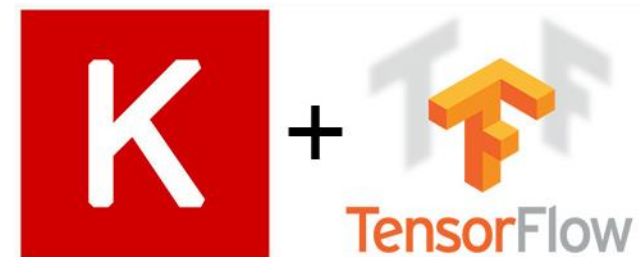


KERAS

- Keras **una biblioteca de código abierto escrita en Python** que facilita la creación de modelos de aprendizaje profundo. Es ampliamente utilizada en la comunidad de investigación y en aplicaciones industriales debido a su simplicidad y capacidad para prototipar modelos rápidamente.

Características

- Prototipado rápido del modelo.
- De alto nivel (programación a nivel de capa)
 - Keras todavía se considera una API de alto nivel. Permite a los usuarios construir modelos a través de una abstracción clara de capas, lo que simplifica el proceso de construcción de arquitecturas complejas.
- Desde 2019, Keras se ha integrado completamente en TensorFlow.
 - Keras es ahora la API oficial de alto nivel de TensorFlow, y su desarrollo independiente con otros backends ha quedado obsoleto



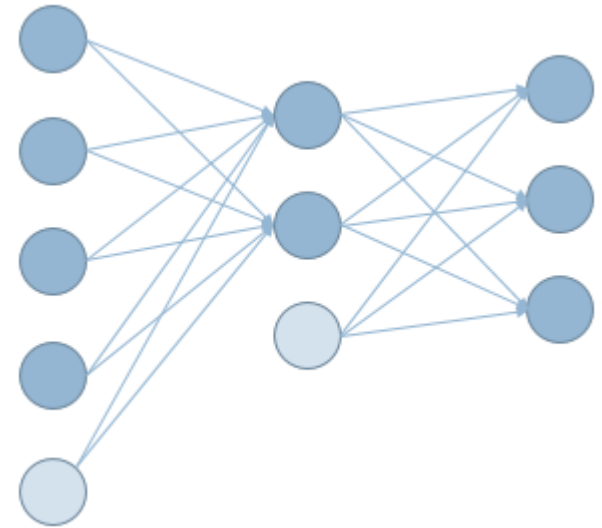
CONSTRUCCIÓN DEL MODELO

```
from keras.models import Sequential
```

```
from keras.layers import Dense
```

Crear un modelo de capas secuenciales

```
model=Sequential()
```



CONSTRUCCIÓN DEL MODELO

```
from keras.models import Sequential
```

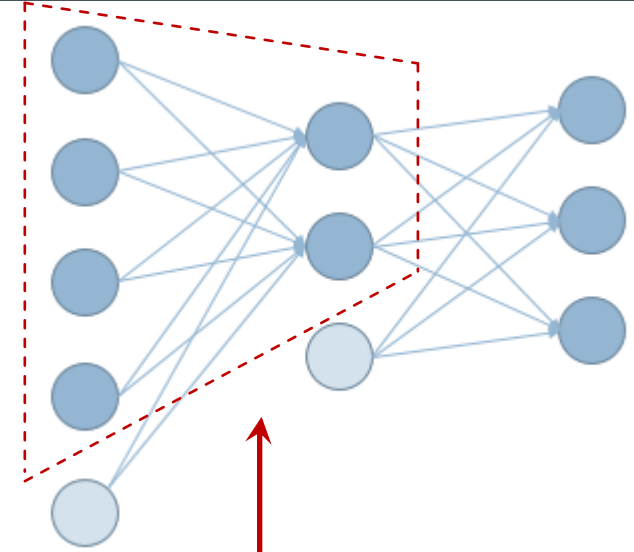
```
from keras.layers import Dense
```

Crear un modelo de capas secuenciales

```
model=Sequential()
```

Agregar las capas al modelo

```
model.add(Dense(2, input_shape=[4], activation='tanh'))
```



CONSTRUCCIÓN DEL MODELO

```
from keras.models import Sequential  
from keras.layers import Dense
```

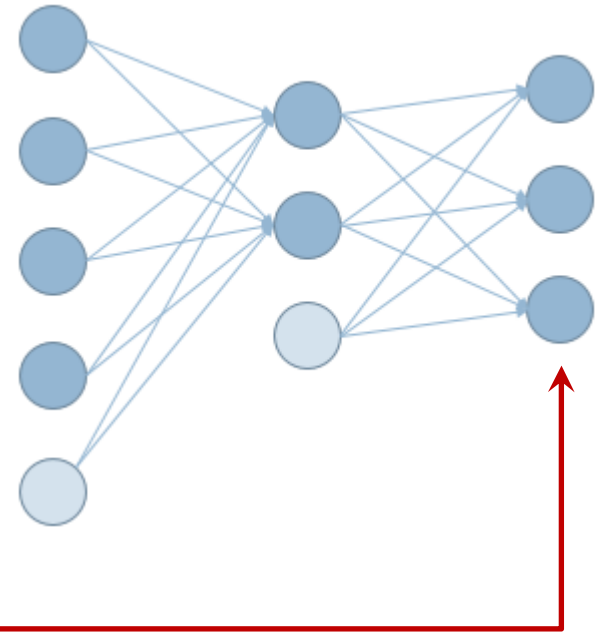
Crear un modelo de capas secuenciales

```
model=Sequential()
```

Agregar las capas al modelo

```
model.add(Dense(2, input_shape=[4], activation='tanh'))
```

```
model.add(Dense(3, activation='sigmoid'))
```



CONSTRUCCIÓN DEL MODELO

```
from keras.models import Sequential  
from keras.layers import Dense
```

Crear un modelo de capas secuenciales

```
model=Sequential()
```

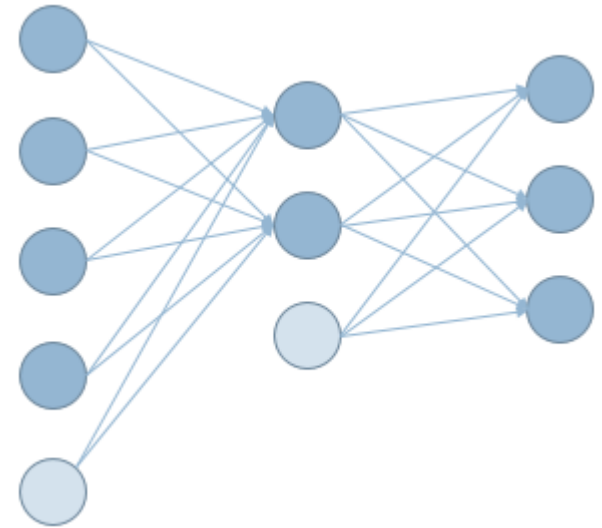
Agregar las capas al modelo

```
model.add(Dense(2, input_shape=[4], activation='tanh'))
```

```
model.add(Dense(3, activation='sigmoid'))
```

Imprimir un resumen del modelo

```
model.summary()
```



Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 2)	10
dense_2 (Dense)	(None, 3)	9
Total params: 19		
Trainable params: 19		
Non-trainable params: 0		

CONSTRUCCIÓN DEL MODELO

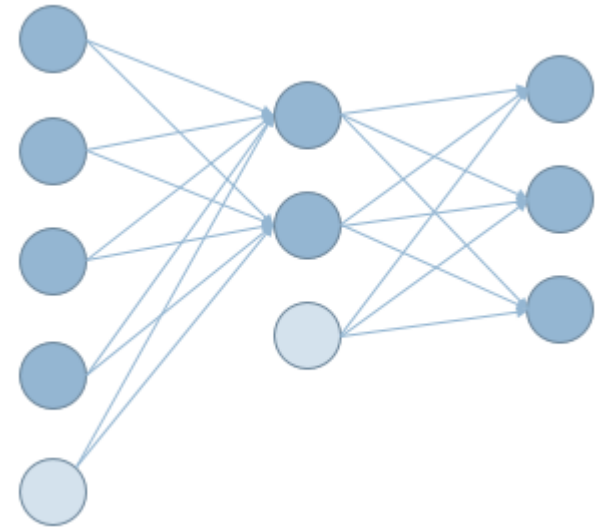
```
from keras.models import Sequential
from keras.layers import Dense, Input
```

Crear un modelo de capas secuenciales

```
model=Sequential()
model.add(Input(shape=(4,)))
model.add(Dense(2, activation='tanh'))
model.add(Dense(3, activation='sigmoid'))
```

Imprimir un resumen del modelo

```
model.summary()
```



Se puede indicar la capa de entrada de manera separada

RESOLUCIÓN DE UNA TAREA DE CLASIFICACIÓN

- Conjunto de datos etiquetados (aprendizaje supervisado)
- Definición de la arquitectura de la red
 - Número de capas y tamaño de cada una
 - Función de activación a usar en cada capa
- Entrenamiento
 - Función de costo
 - Técnica de optimización para reducir el error
- Evaluar el modelo



CONFIGURACIÓN PARA ENTRENAMIENTO

```
from keras.models import Sequential  
from keras.layers import Dense
```

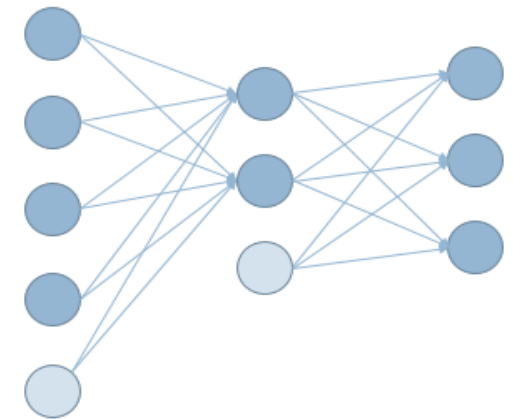
```
model=Sequential()  
model.add(Dense(2, input_shape=[4], activation='tanh'))  
model.add(Dense(3, activation='sigmoid'))
```

Configuración para entrenamiento

```
model.compile(optimizer='sgd', loss='mse', metrics='accuracy')
```

*Descenso de gradiente
estocástico*

*Error Cuadrático
Medio*



Keras_Iris.ipynb

CONFIGURACIÓN PARA ENTRENAMIENTO

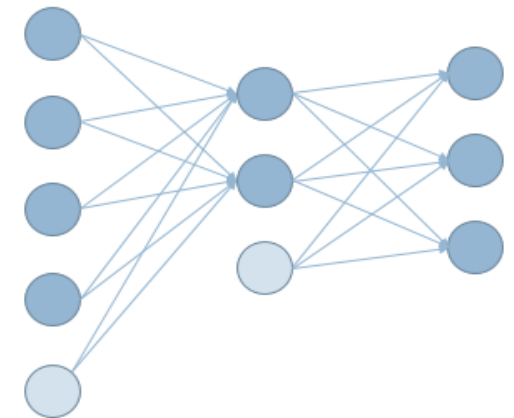
```
from keras.optimizers import SGD
from keras.models import Sequential
from keras.layers import Dense
```

```
model=Sequential()
model.add(Dense(2, input_shape=[4], activation='tanh'))
model.add(Dense(3, activation='sigmoid'))
```

Configuración para entrenamiento

```
model.compile(optimizer=SGD(learning_rate=0.1), loss='mse', metrics='accuracy')
```

Tasa de aprendizaje



Keras_Iris.ipynb

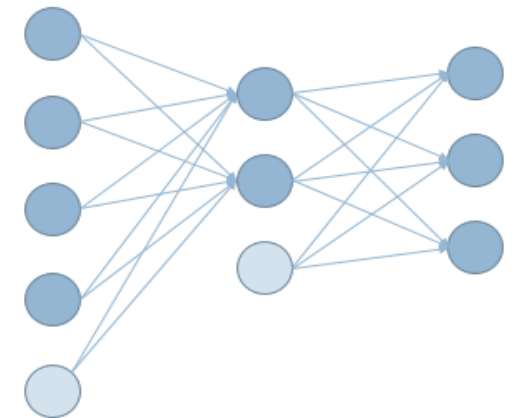
CONFIGURACIÓN PARA ENTRENAMIENTO

```
from keras.models import Sequential  
from keras.layers import Dense
```

```
model=Sequential()  
model.add(Dense(2, input_shape=[4], activation='tanh'))  
model.add(Dense(3, activation='softmax'))
```

Configuración para entrenamiento

```
model.compile(loss='categorical_crossentropy', optimizer='sgd',  
              metrics=['accuracy'])
```



Keras_Iris.ipynb

CARGA DE DATOS

`X, T = cargar_datos()`

`Y = keras.utils.to_categorical(T)`

T debe ser un vector numérico. Puede usar lo siguiente para convertirlo de ser necesario:

```
from sklearn import preprocessing
encoder = preprocessing.LabelEncoder()
T = encoder.fit_transform(T)
```

X → Conjunto de ejemplos de entrada

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4

Y → Rtas esperadas para c/neurona de la capa de salida

	0	1	2
0	1	0	0
1	0	1	0
2	0	0	1
3	0	0	1
4	0	1	0
5	0	0	1

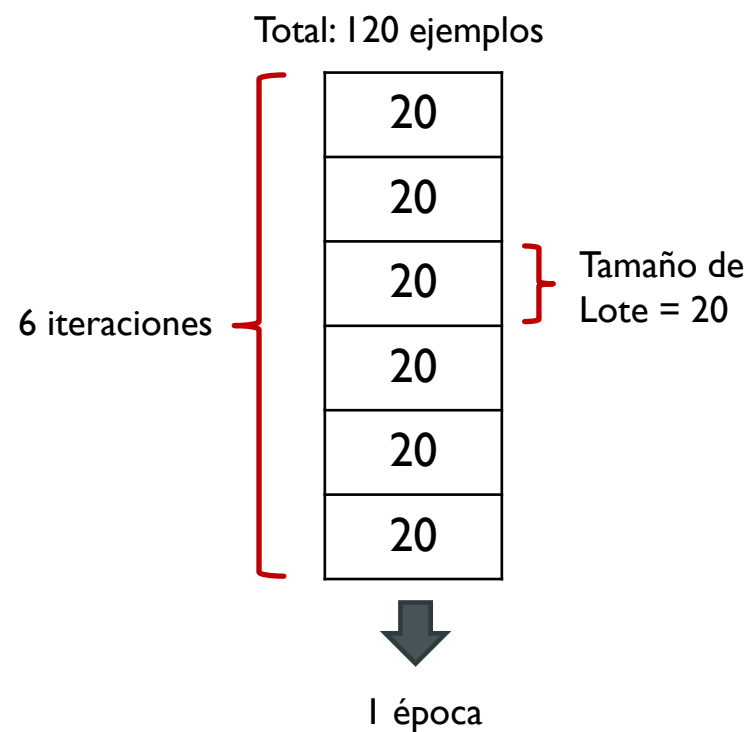
X e Y son matrices de numpy

ENTRENAMIENTO DEL MODELO

`X,Y = cargar_datos()` **# X e Y son matrices de numpy**

Entrenar el modelo

`model.fit(X,Y, epochs=100, batch_size=20)`



PREDICCIÓN DEL MODELO

`X,Y = cargar_datos()` *# X e Y son matrices de numpy*

Entrenar el modelo

`model.fit(X,Y, epochs=100, batch_size=20)`

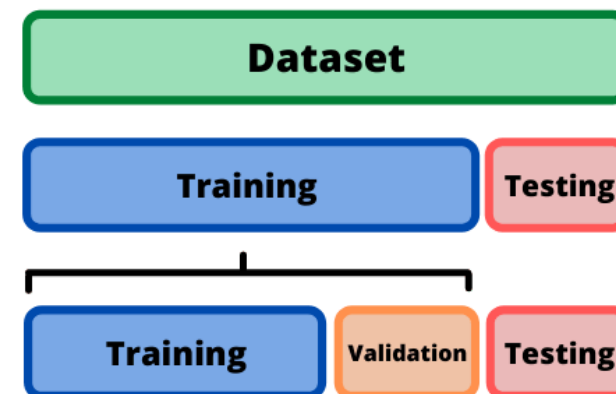
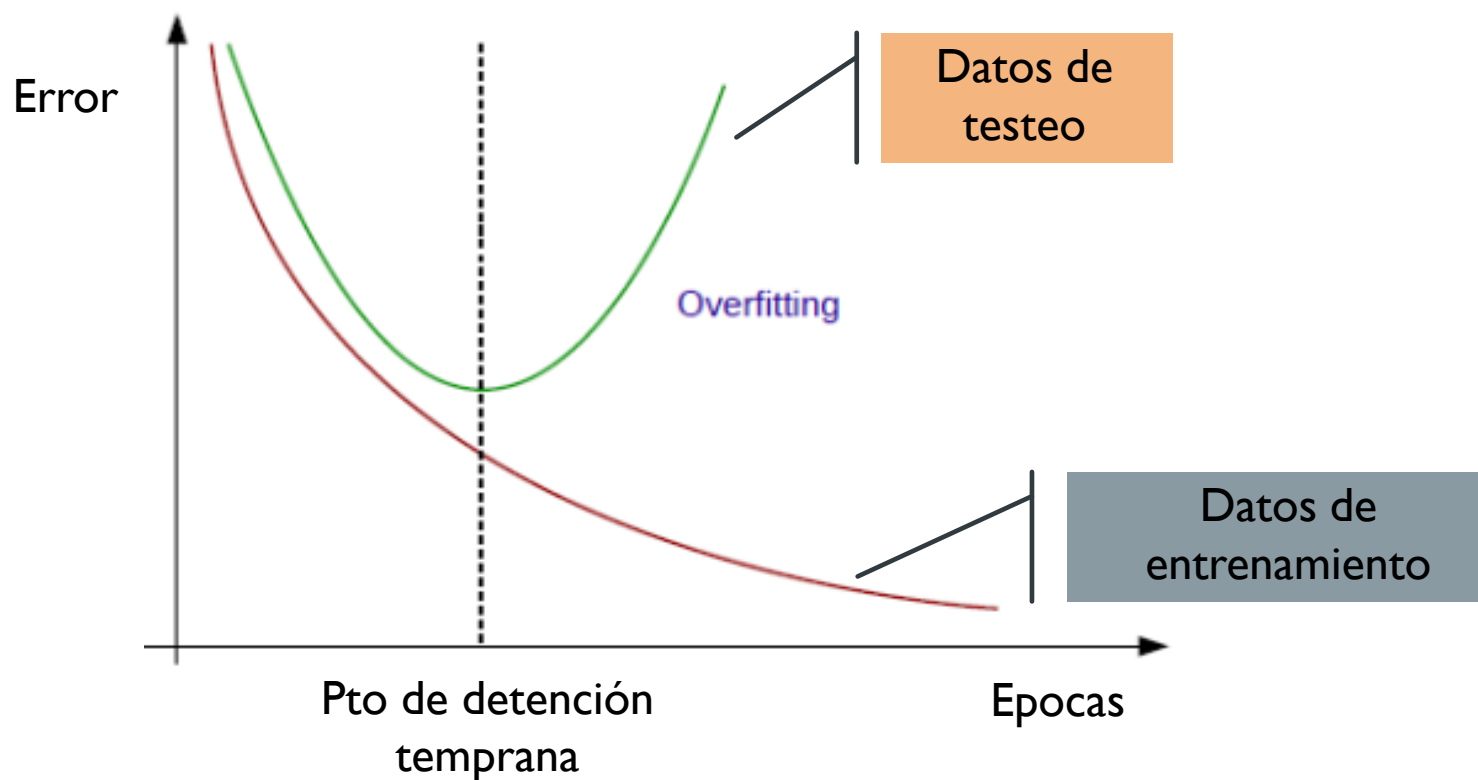
predecir la salida del modelo

`Y_pred = model.predict(X)`

Y_pred tiene las mismas
dimensiones que **Y**

	0	1	2
0	0.967722	0.189344	0.00421873
1	0.0372113	0.510963	0.346058
2	0.00325751	0.261545	0.917956
3	0.00823823	0.319694	0.795647
4	0.0717264	0.611822	0.171516
5	0.0134856	0.482814	0.59486

PARADA TEMPRANA (EARLY-STOPPING)



PARADA TEMPRANA (EARLY-STOPPING)

```
from keras.callbacks import EarlyStopping
```

```
model = ...
```

```
model.compile( ... )
```

```
es = EarlyStopping(monitor='val_accuracy', patience=30, min_delta=0.0001)
```

```
H = model.fit(x = X_train, y = Y_train, epochs=4000, batch_size = 20,  
             validation_data = (X_test, Y_test), callbacks=[es])
```



```
print("Épocas = %d" % es.stopped_epoch)
```

Un **callback** es un objeto que puede realizar acciones en varias etapas del entrenamiento (por ejemplo, al inicio o final de una época, antes o después de un lote)

PARADA TEMPRANA (EARLY-STOPPING)

```
from keras.callbacks import EarlyStopping  
model = ...  
model.compile( ... )
```


EarlyStopping es un callback que detiene el entrenamiento de un modelo de forma anticipada si no hay mejoras en un criterio específico

```
es = EarlyStopping(monitor='val_accuracy', patience=30, min_delta=0.0001)  
H = model.fit(x = X_train, y = Y_train, epochs=4000, batch_size = 20,  
              validation_data = (X_test, Y_test), callbacks=[es])  
  
print("Épocas = %d" % es.stopped_epoch)
```

min_delta=0.0001 significa que se considera una mejora si el incremento de la precisión es mayor que 0.0001

Keras_Iris.ipynb

RESOLUCIÓN DE UNA TAREA DE CLASIFICACIÓN

- Conjunto de datos etiquetados (aprendizaje supervisado)
- Definición de la arquitectura de la red
 - Número de capas y tamaño de cada una
 - Función de activación a usar en cada capa
- Entrenamiento
 - Función de costo
 - Técnica de optimización para reducir el error
- Evaluar el modelo 

ERROR DEL MODELO

```
X,Y = cargar_datos() # X e Y son matrices de numpy
```

```
# Entrenar el modelo
```

```
model.fit(X,Y, epochs=100, batch_size=20)
```

```
# predecir la salida del modelo
```

```
Y_pred = model.predict(X)
```

```
# Calcular el error del modelo
```

```
score = model.evaluate(X_train,Y_trainB)
```

```
print('Error :', score[0])
```

```
print('Accuracy:', score[1])
```

*Muestra el valor de la función de Costo
y la precisión del modelo al finalizar el
entrenamiento*

MATRIZ DE CONFUSIÓN

	Predice Clase 1	Predice Clase 2	Recall
True Clase 1	A	B	$A/(A+B)$
True Clase 2	C	D	$D/(C+D)$
Precision	$A/(A+C)$	$D/(B+D)$	$(A+D)/(A+B+C+D)$ accuracy

- Los **aciertos** del modelo están sobre la **diagonal** de la matriz.
- **Precision**: la proporción de **predicciones correctas** sobre **una clase**.
- **Recall**: la proporción de **ejemplos** de **una clase** que son **correctamente clasificados**.
- **Accuracy**: la performance general del modelo, sobre **todas las clases**. Es la cantidad de **aciertos** sobre el **total** de ejemplos.

MÉTRICAS

```
report = metrics.classification_report(Y_true, Y_pred, target_names=etiquetas)
```

```
print("Training metrics:\n%s" % report)
```

```
MM = metrics.confusion_matrix(Y_true, Y_pred)
```

```
print("Confusion matrix:\n%s" % MM)
```

$$f1 - score = 2 * \frac{precision * recall}{precision + recall}$$

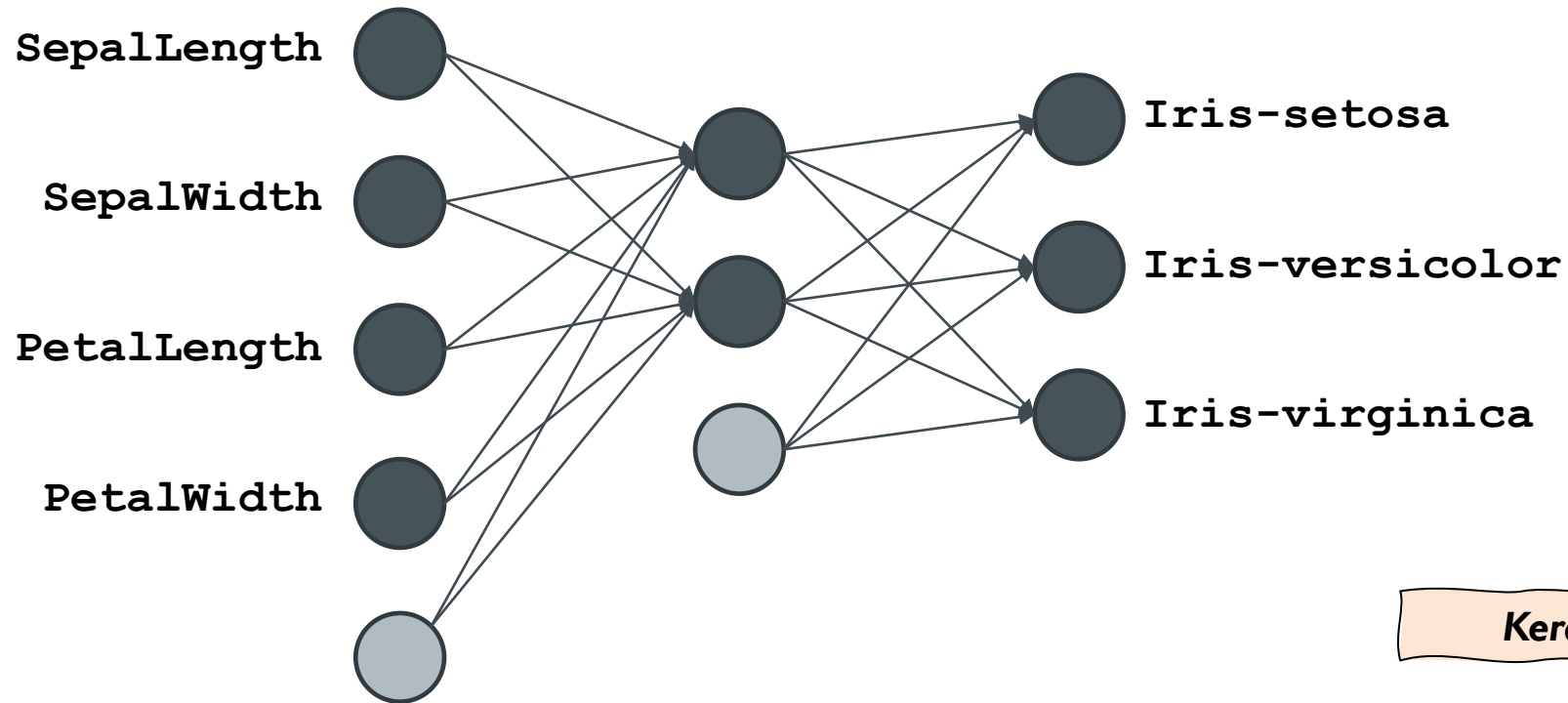
Training metrics:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	42
Iris-versicolor	0.97	0.88	0.92	41
Iris-virginica	0.88	0.97	0.92	37
accuracy			0.95	120
macro avg	0.95	0.95	0.95	120
weighted avg	0.95	0.95	0.95	120

Confusion matrix:

```
[[42  0  0]
 [ 0 36  5]
 [ 0  1 36]]
```

EJEMPLO: CLASIFICACIÓN DE FLORES DE IRIS



Keras_Iris.ipynb

RECONOCEDOR DE DÍGITOS ESCRITOS A MANO

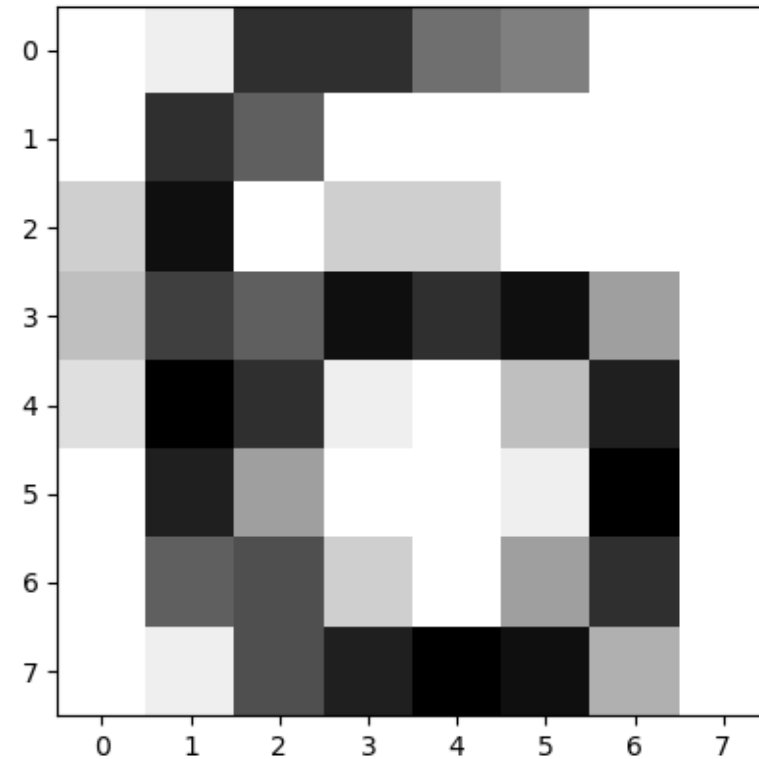
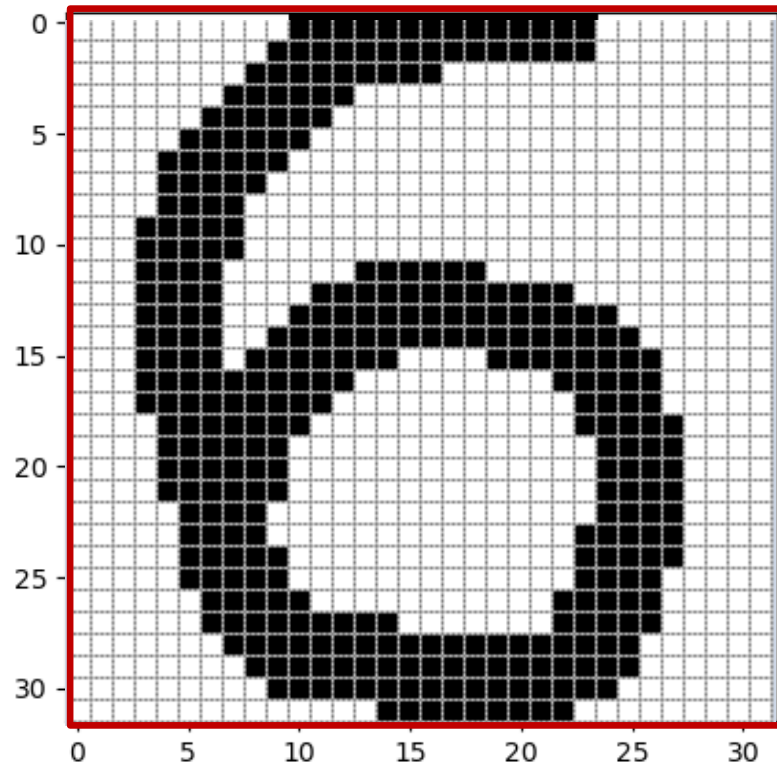
- Se desea entrenar un multiperceptrón para reconocer dígitos escritos a mano. Para ello se dispone de los mapas de bits correspondientes a 5620 dígitos escritos a mano.
- Los primeros 3823 fueron escritos por 30 personas diferentes y deben ser usado para el entrenamiento.
- El desempeño de la red será probado con los 1797 dígitos restantes que fueron escritos por otras 13 personas.



<https://archive.ics.uci.edu/ml/datasets/optical+recognition+of+handwritten+digits>

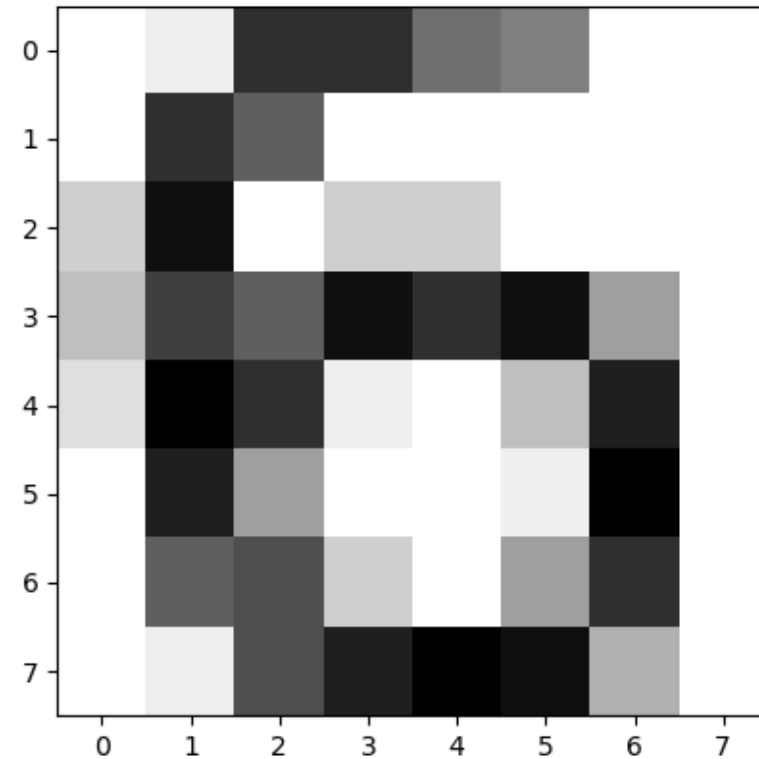
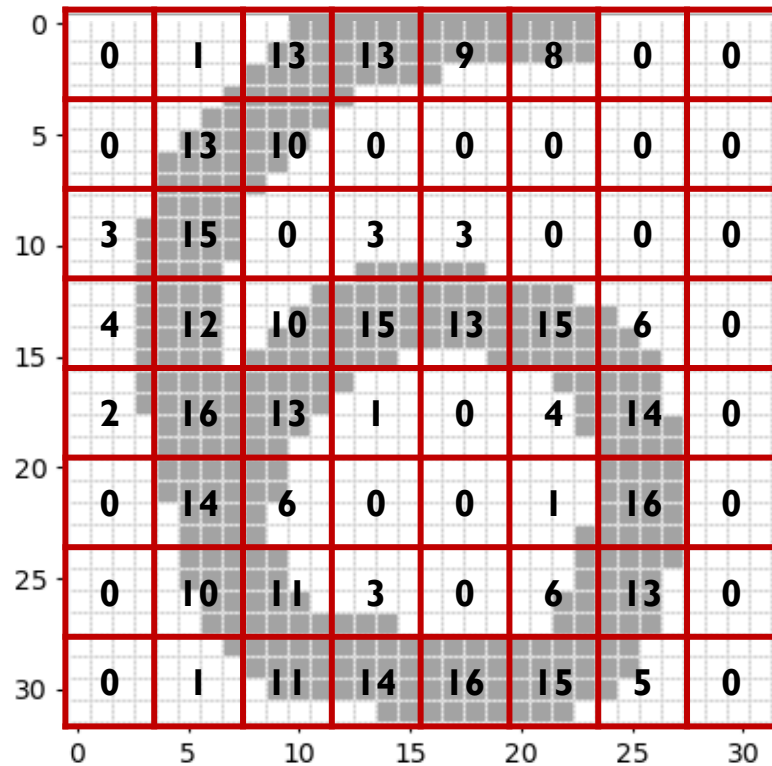
DÍGITOS

- Cada dígito está representado por una matriz numérica de 8x8



DÍGITOS

- Cada dígito está representado por una matriz numérica de 8x8



PREDICCIÓN DE LA RN

```
Y_pred = model.predict(X_test)
Y_pred2 = np.argmax(Y_pred,axis=1)
Y_true = np.argmax(Y_test,axis=1)
print("%% aciertos X_train : %.3f" % metrics.accuracy_score(Y_true, Y_pred2))
```

```
57/57 ————— 0s 1ms/step
% aciertos X_train : 0.961
```

```
report = metrics.classification_report(Y_true, Y_pred2)
print("Dígitos de testeo :\n%s" % report)
```

```
Dígitos de testeo :
              precision    recall  f1-score   support

0               0.99        0.98        0.99         178
1               0.93        0.99        0.96         182
2               0.98        0.95        0.97         177
3               0.99        0.96        0.97         183
4               0.98        0.98        0.98         181
5               0.91        0.98        0.94         182
6               1.00        0.99        0.99         181
7               0.98        0.92        0.95         179
8               0.96        0.89        0.92         174
9               0.90        0.97        0.94         180

 accuracy                   0.96         1797
 macro avg                  0.96         1797
 weighted avg               0.96         1797
```

Keras_DIGITOS.ipynb