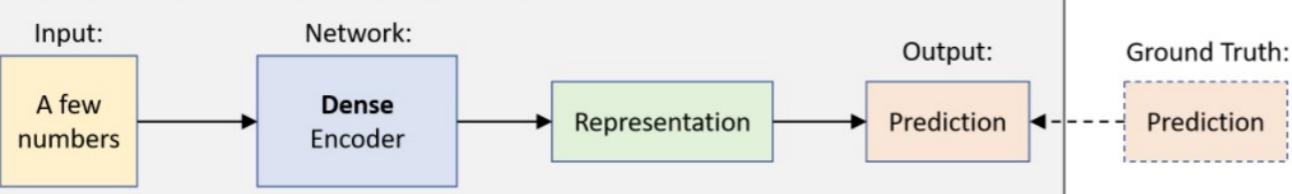


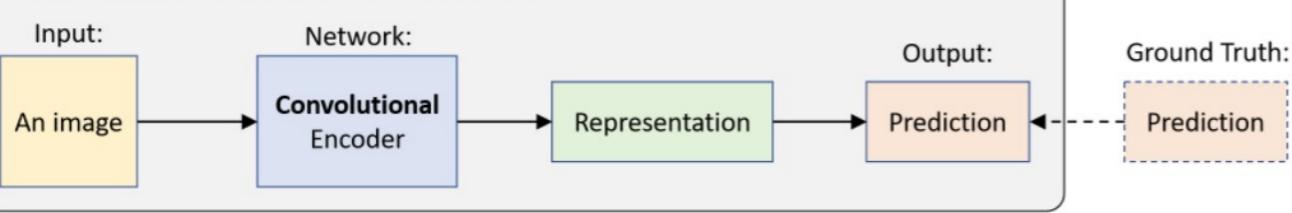
# Redes neuronales

## Supervised Learning

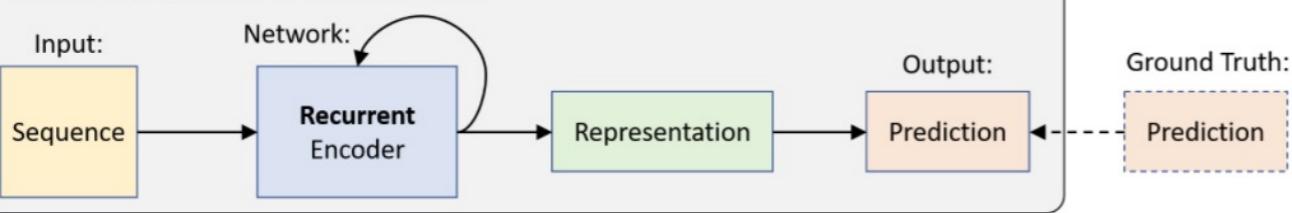
### 1. Feed Forward Neural Networks



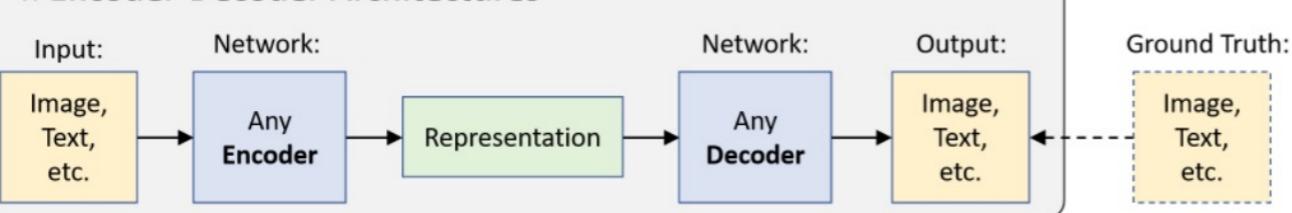
### 2. Convolutional Neural Networks



### 3. Recurrent Neural Networks

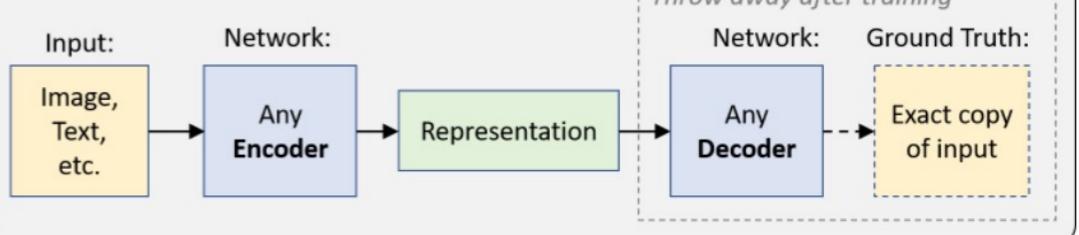


### 4. Encoder-Decoder Architectures

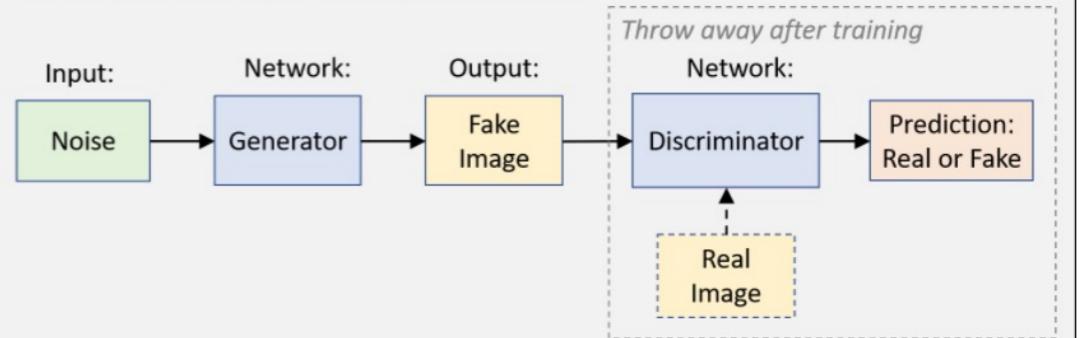


## Unsupervised Learning

### 5. Autoencoder

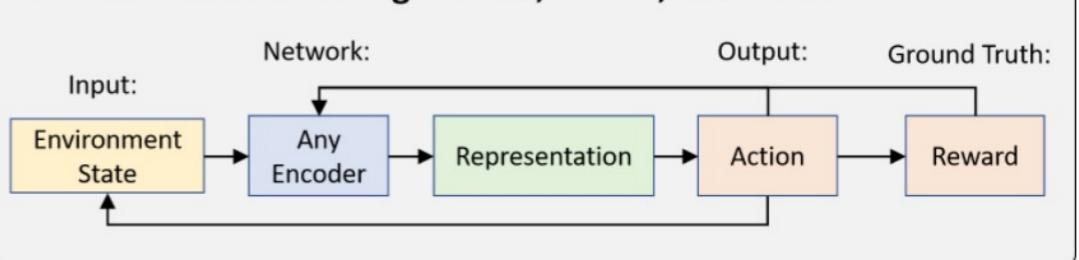


### 6. Generative Adversarial Networks



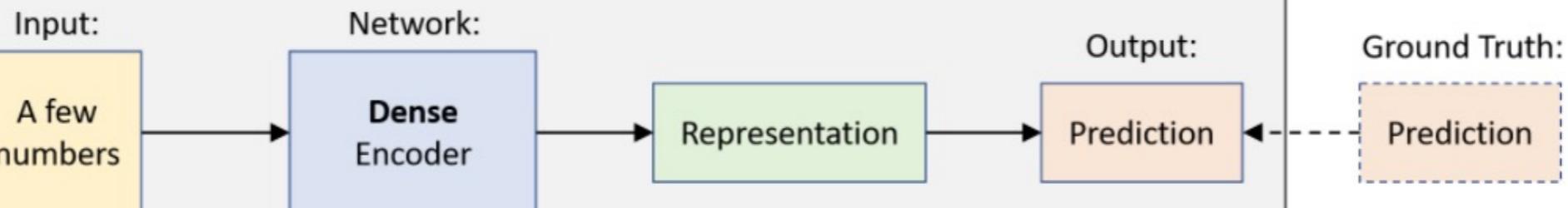
## Reinforcement Learning

### 7. Networks for Learning Actions, Values, and Policies

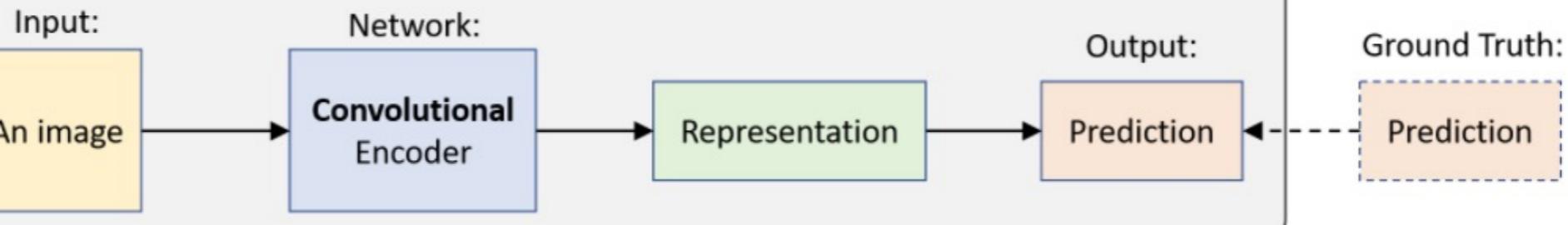


# Supervised Learning

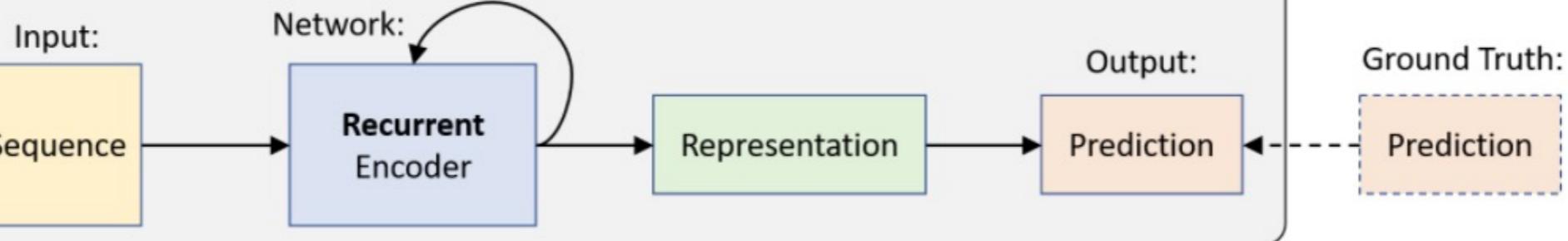
## 1. Feed Forward Neural Networks



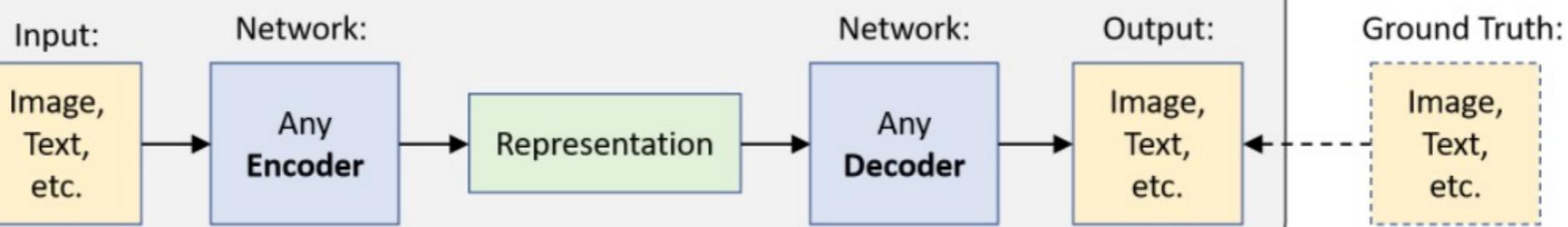
## 2. Convolutional Neural Networks



## 3. Recurrent Neural Networks

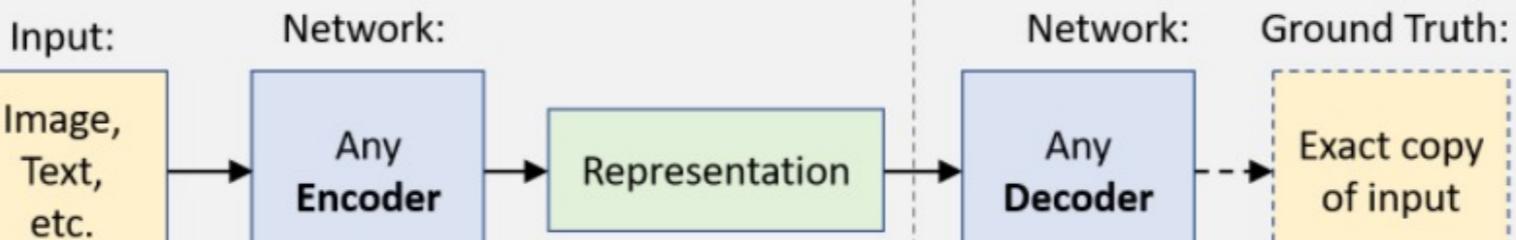


## 4. Encoder-Decoder Architectures

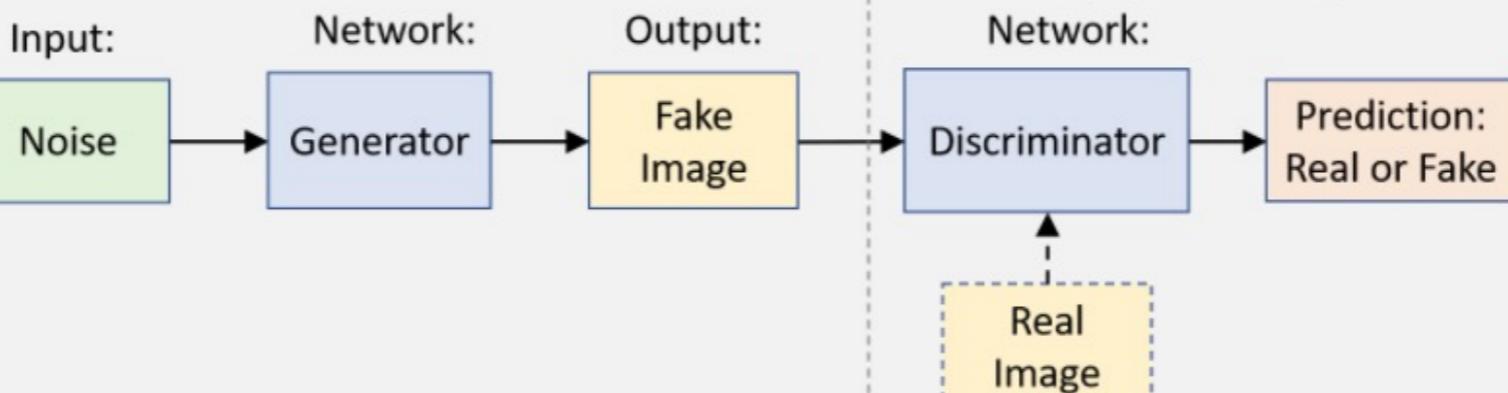


# Unsupervised Learning

## 5. Autoencoder

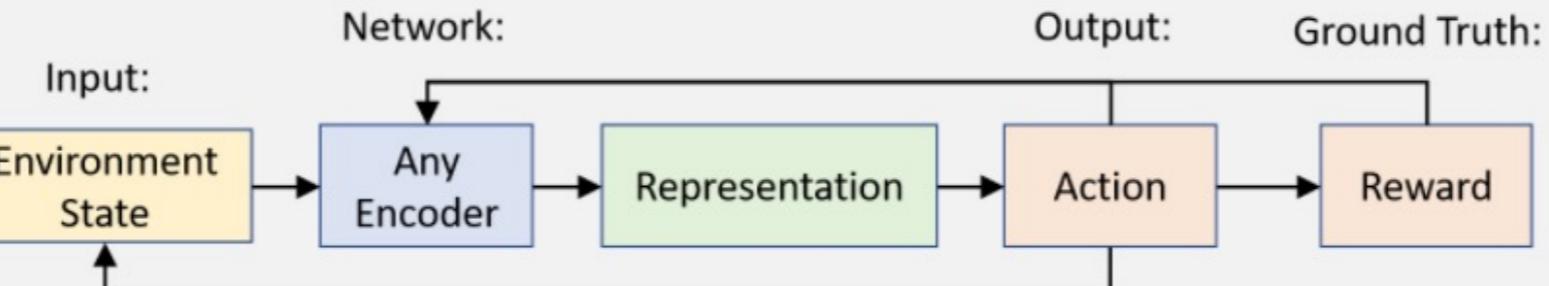


## 6. Generative Adversarial Networks

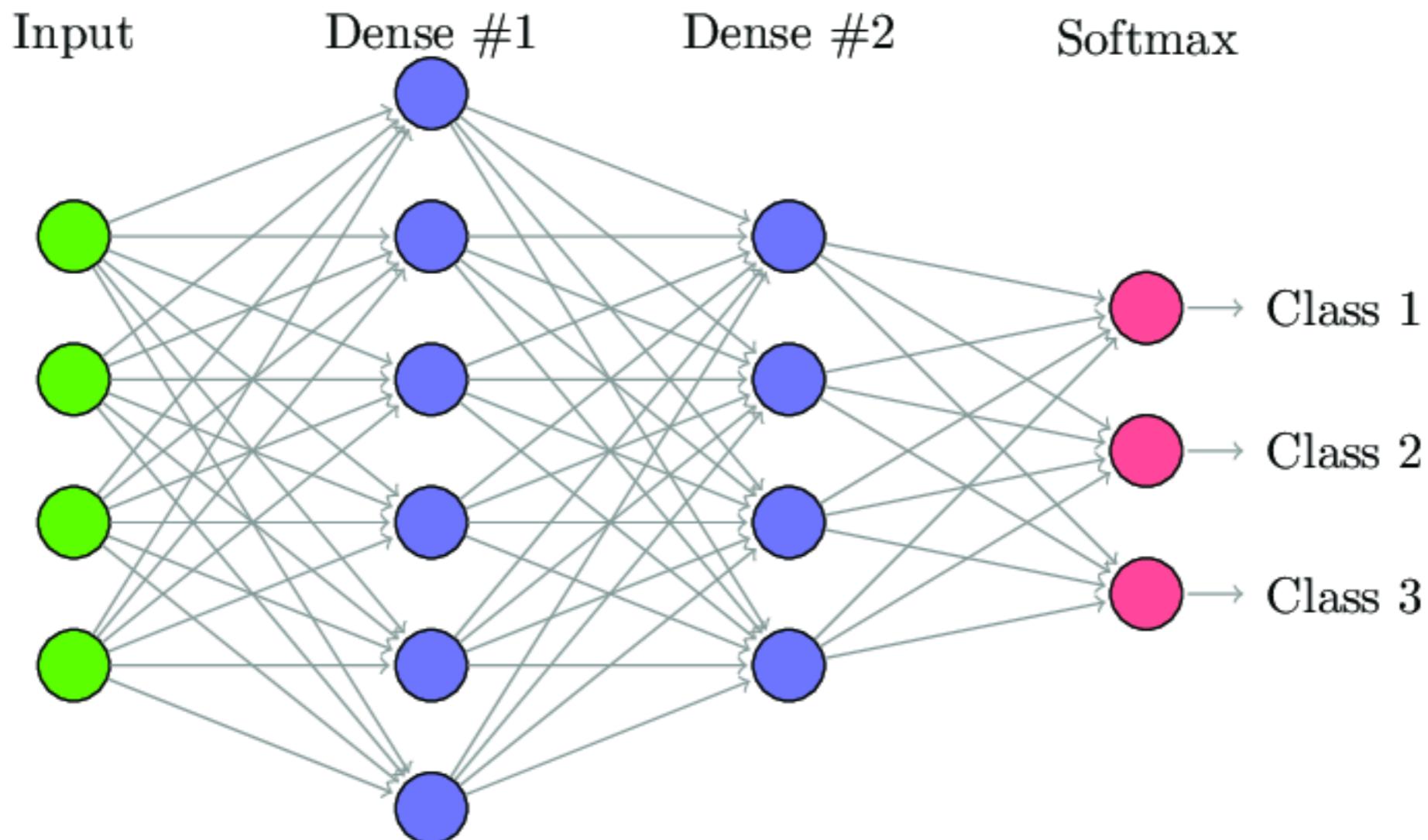


# Reinforcement Learning

## 7. Networks for Learning Actions, Values, and Policies

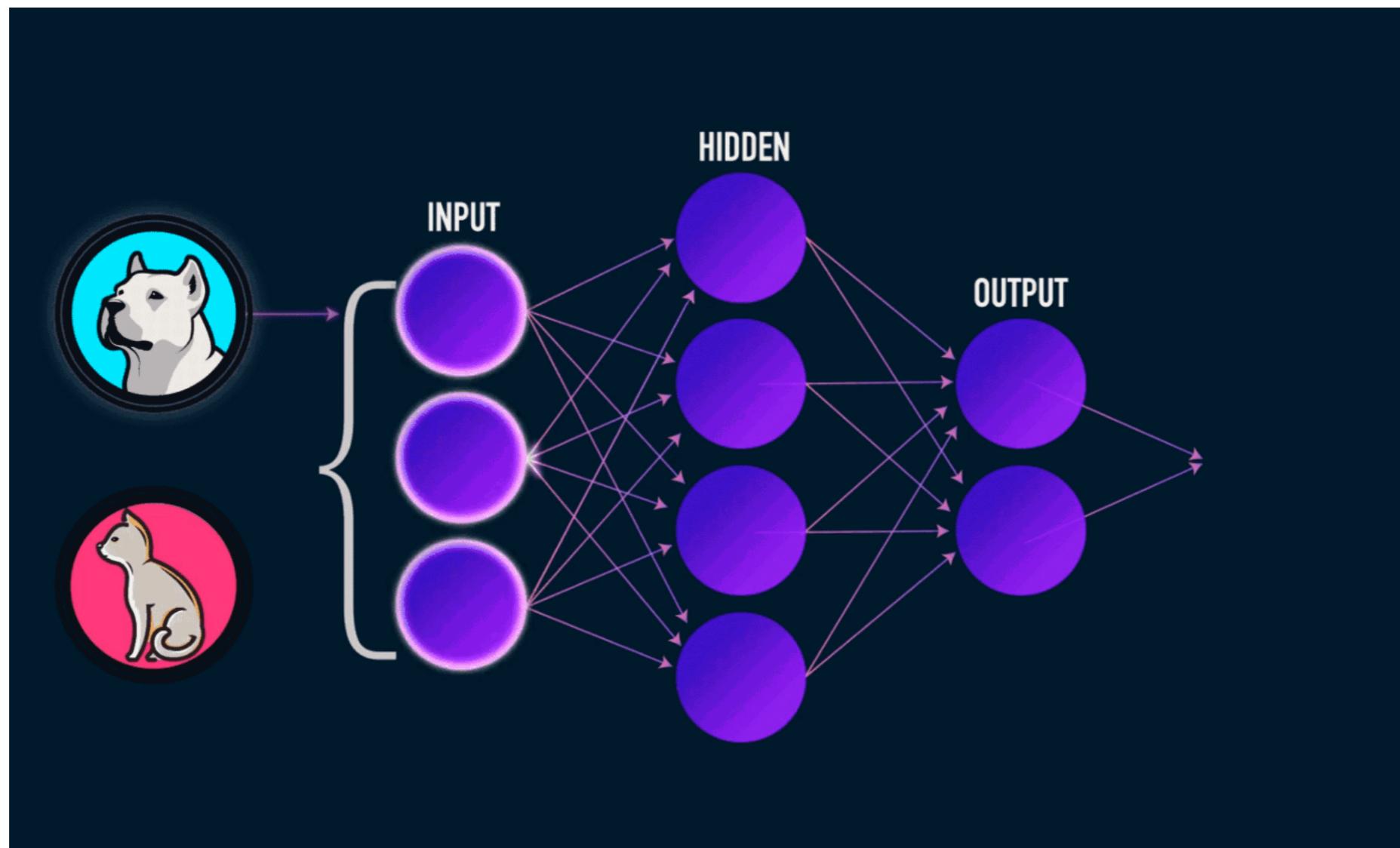


# Redes neuronales totalmente conectadas



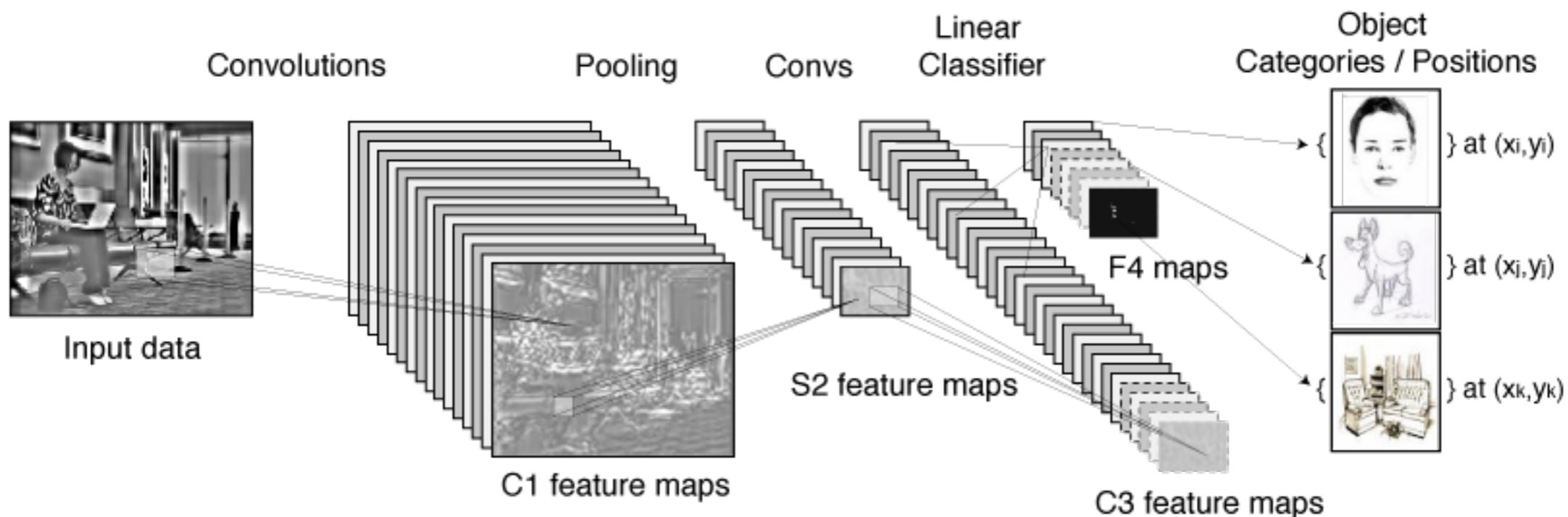
# Redes Neuronales Convolucionales (CNN)

Especializadas en el tratamiento de imágenes mediante la búsqueda de patrones bidimensionales



# Redes Neuronales Convolucionales (CNN)

Especializadas en el tratamiento de imágenes mediante la búsqueda de patrones bidimensionales



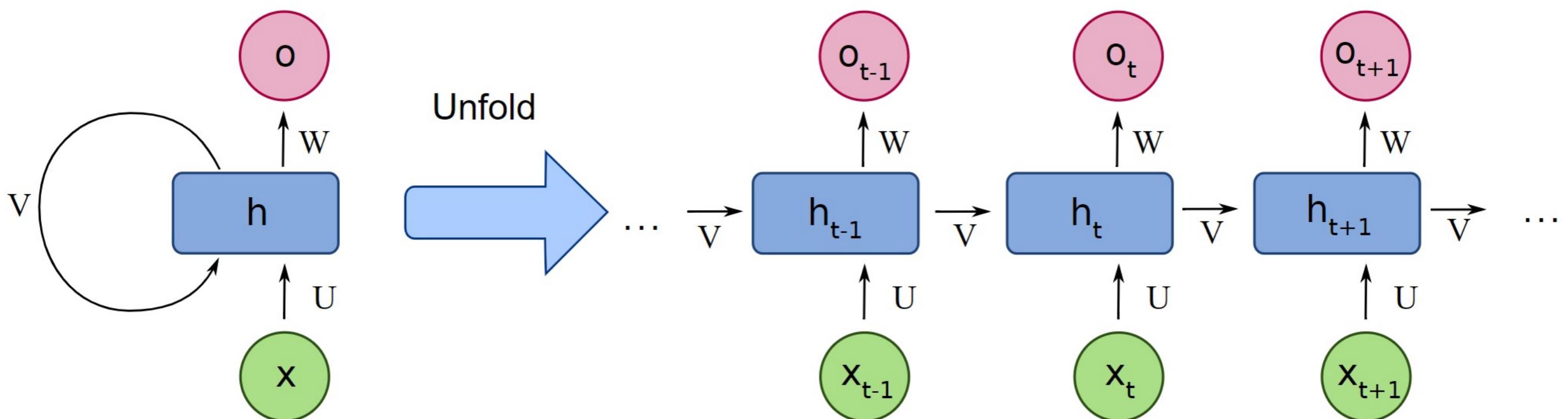
# Redes Neuronales Convolucionales (CNN)

Especializadas en el tratamiento de imágenes  
mediante la búsqueda de patrones bidimensionales



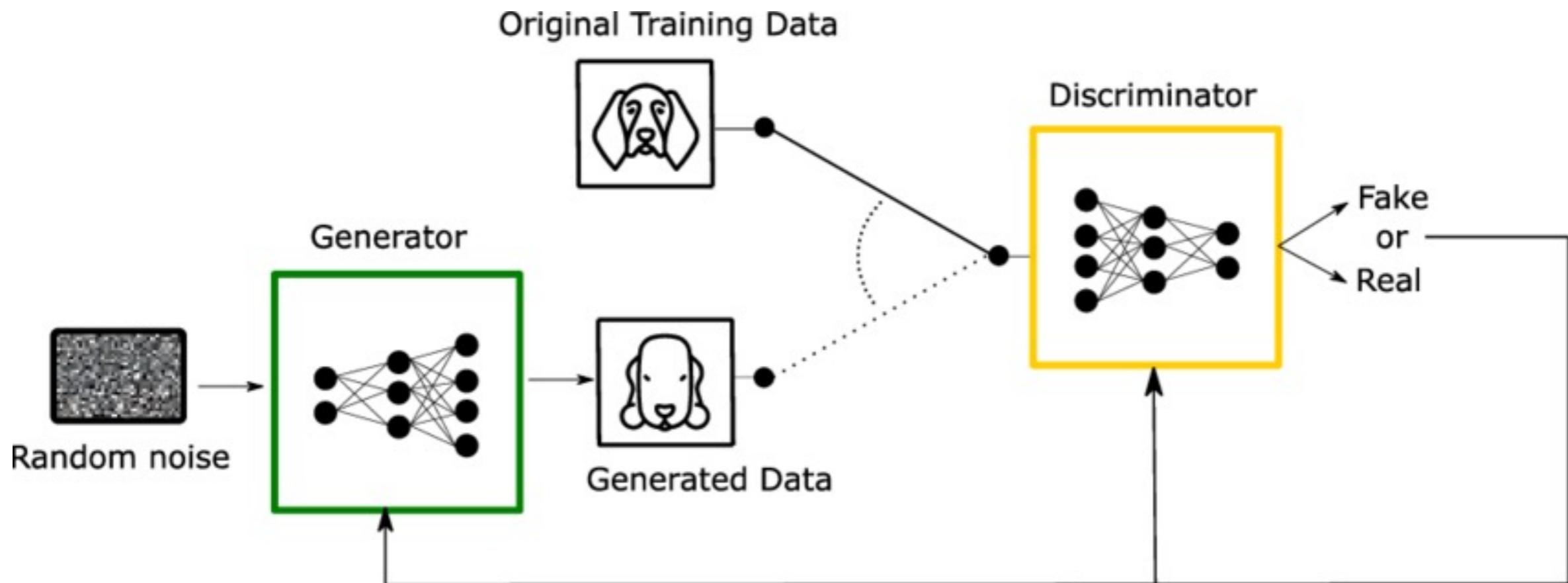
# Redes Neuronales Recurrentes (RNN)

Tratan series ordenadas en el tiempo (procesamiento de voz, texto, series genéticas, etc.).



# Redes Neuronales Generativas Adversarias (GAN)

Generación de datos sintéticos



# Redes Neuronales Generativas Adversarias (GAN)

**Generar ejemplos para conjuntos de datos de imágenes (aumento de datos)**

**Generar fotografías de rostros humanos**

**Genera fotografías realistas**

**Generar personajes de dibujos animados**

**Traducción de imagen a imagen**

**Traducción de texto a imagen**

**Traducción semántica de imagen a foto**

**Generación de vista frontal de cara**

**Genera nuevas poses humanas**

**Fotos a Emojis**

**Edición de fotografía**

**Envejecimiento facial**

**Fusión de fotos**

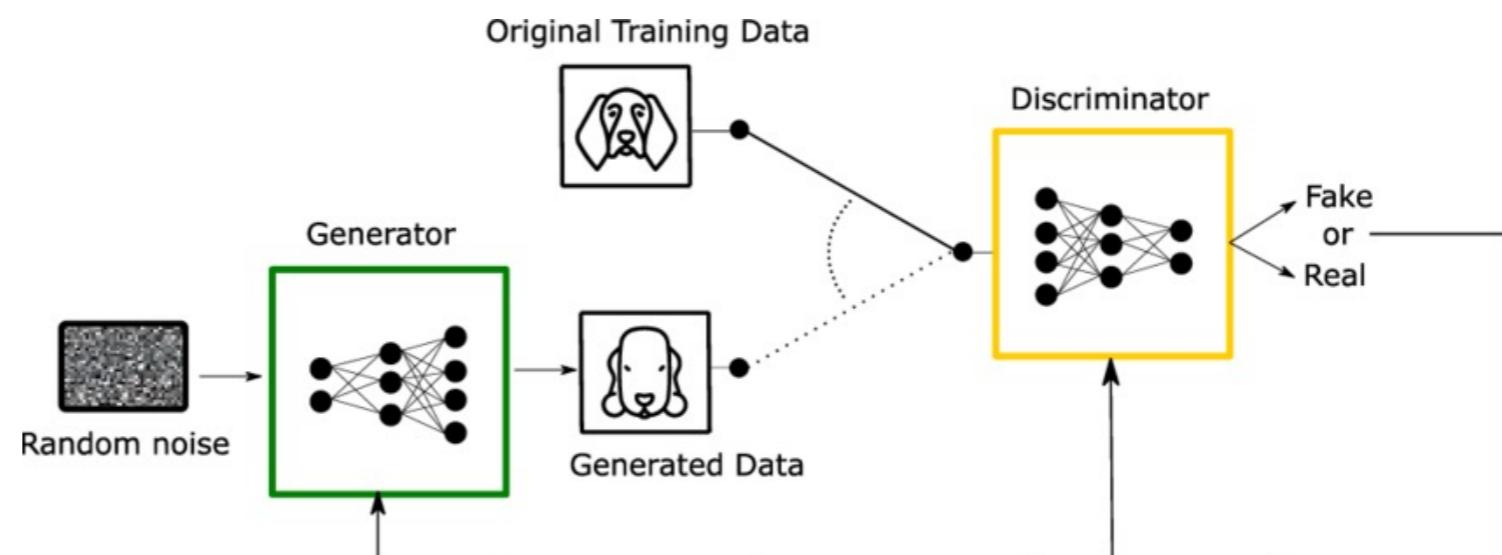
**Súper resolución**

**Repintado de fotos**

**Integración de ropa**

**Predicción de vídeo**

**Generación de objetos 3D**

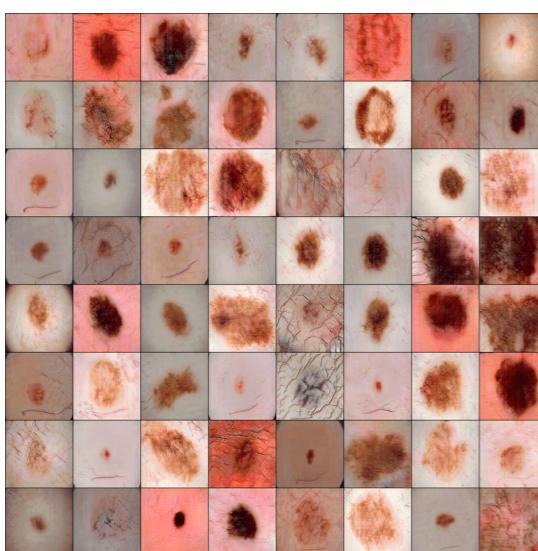


# Redes Neuronales Generativas Adversarias (GAN)

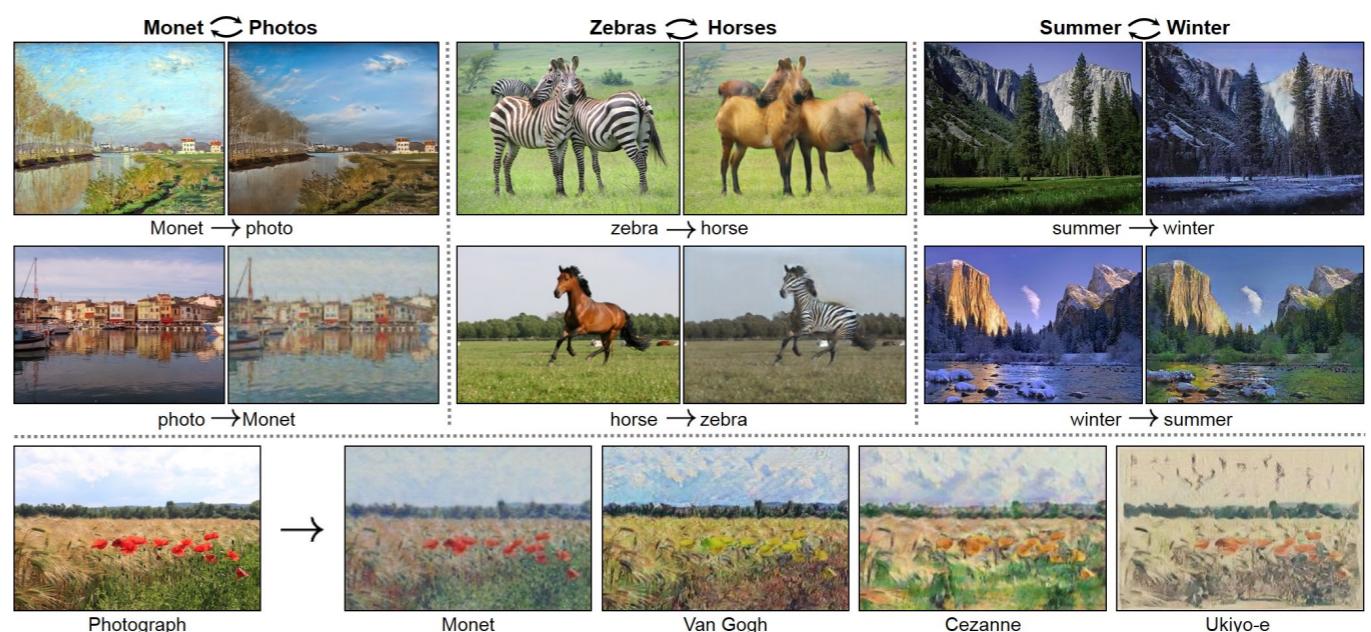
## Generación de datos sintéticos



Bas Uterwijk, who publishes faces "recreated" by machine learning technology from a software called Artbreeder.

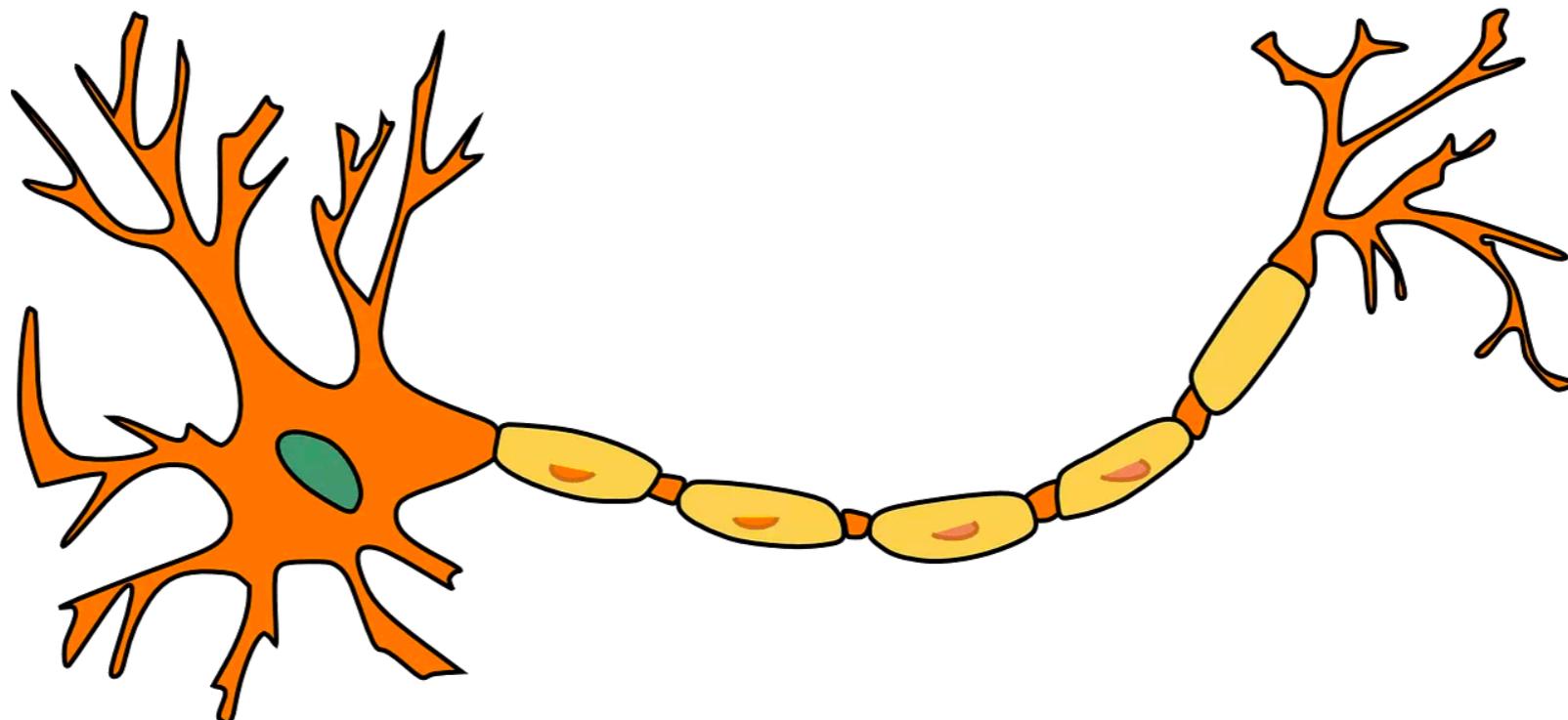


GAN-generated skin lesion samples (a) and their segmentation masks



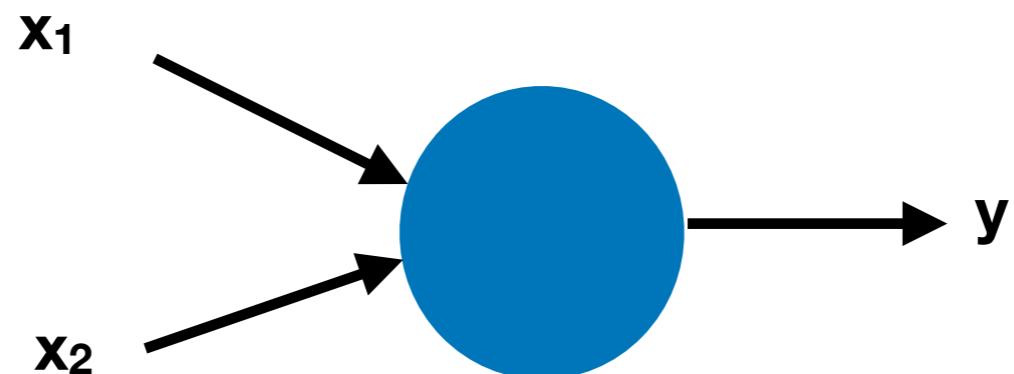
# Red Neuronal Profunda

Neurona



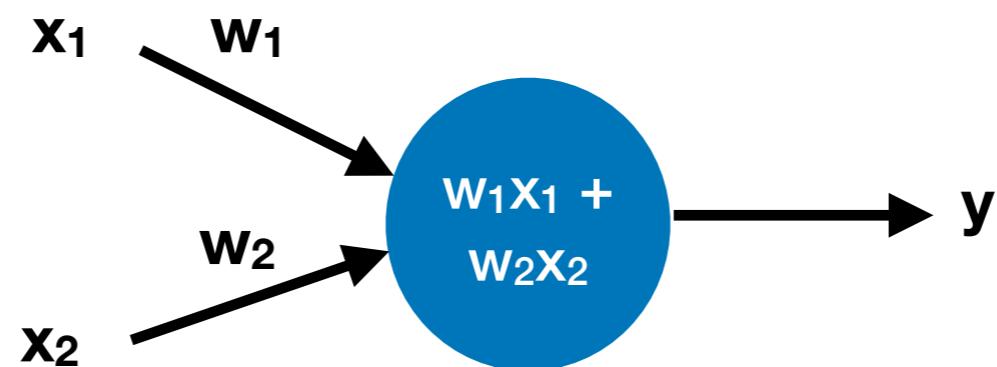
# Red Neuronal Profunda

## Perceptron



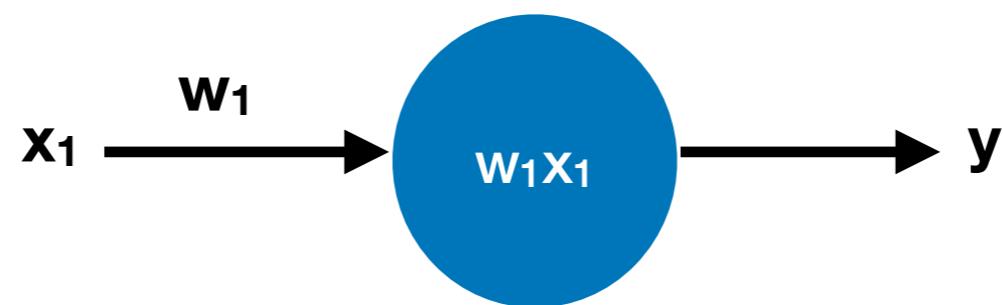
# Red Neuronal Profunda

## Perceptron



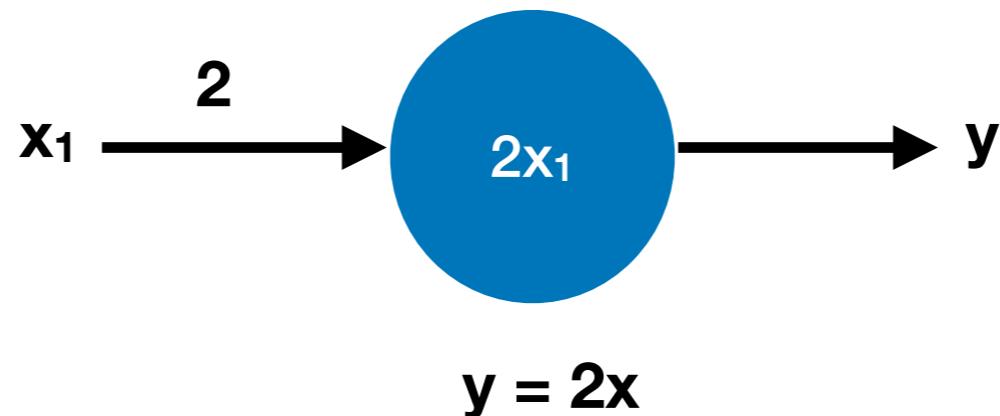
# Red Neuronal Profunda

Perceptron simple  
(una sola variable de entrada)



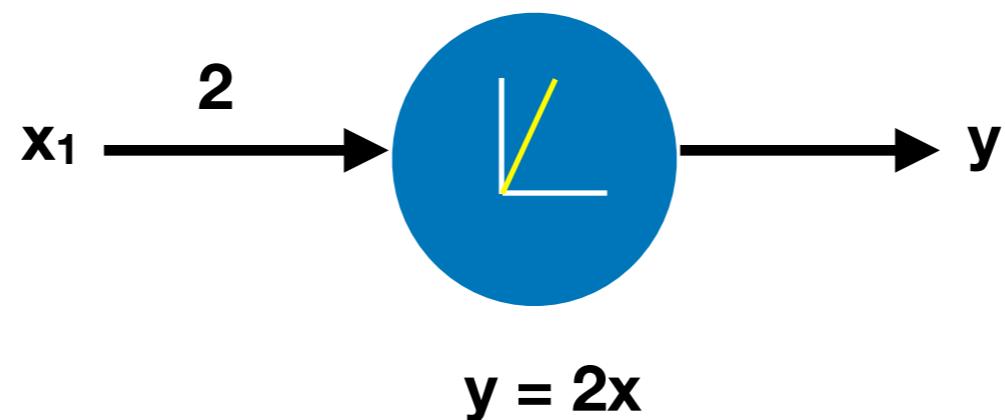
# Red Neuronal Profunda

Perceptron simple  
(una sola variable de entrada)



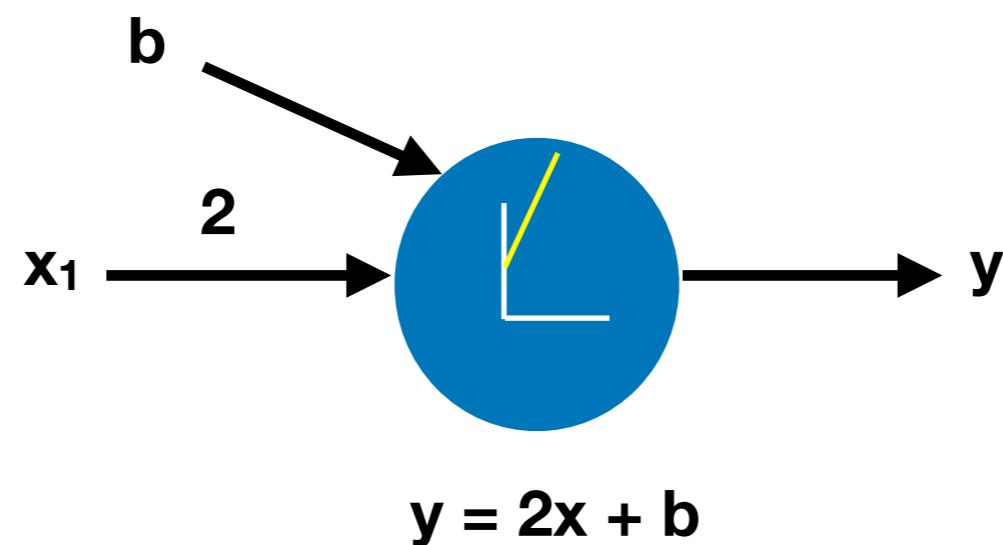
# Red Neuronal Profunda

Perceptrón simple  
(una sola variable de entrada)



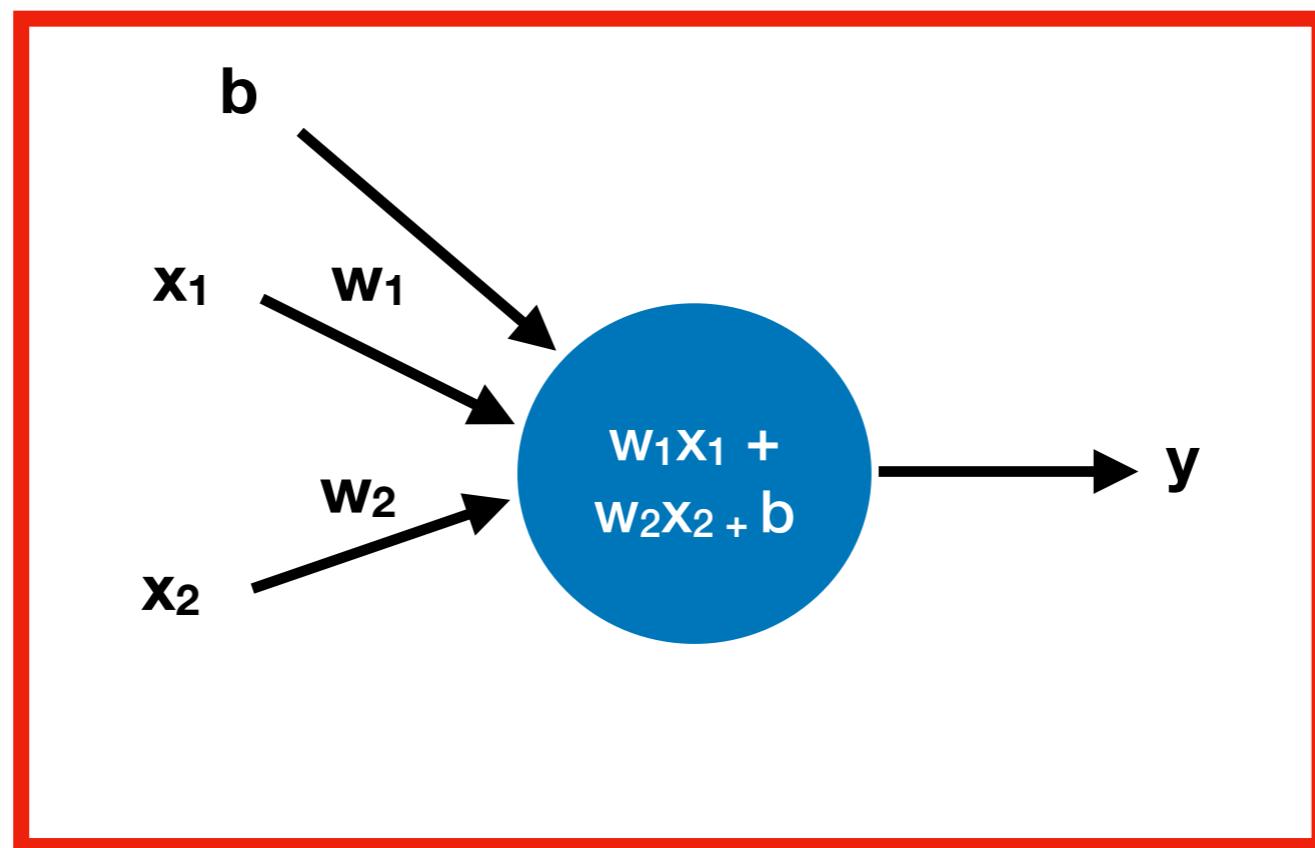
# Red Neuronal Profunda

Perceptrón simple  
(Añadiendo el sesgo)



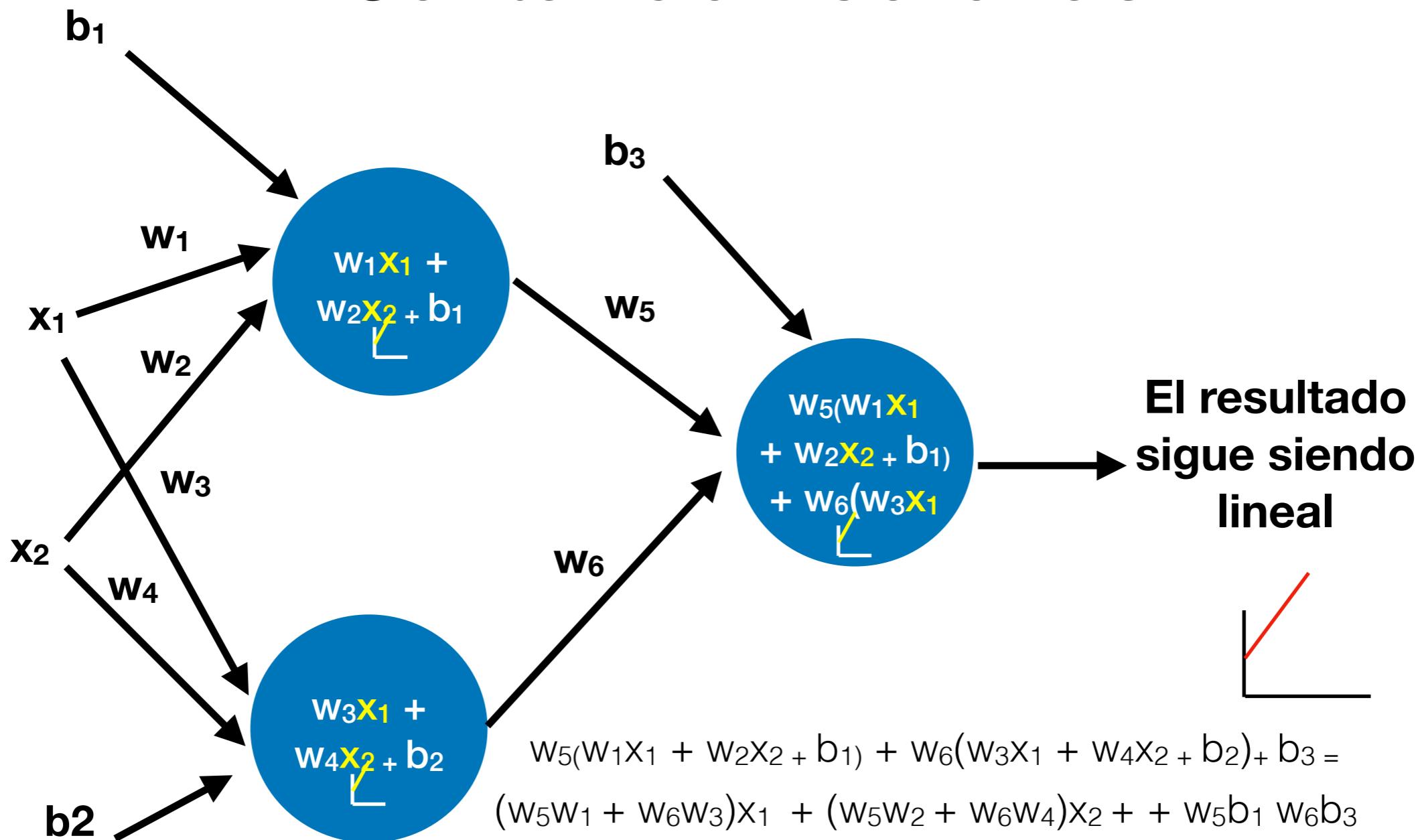
# Red Neuronal Profunda

Perceptron  
(Con dos variables)



# Red Neuronal Profunda

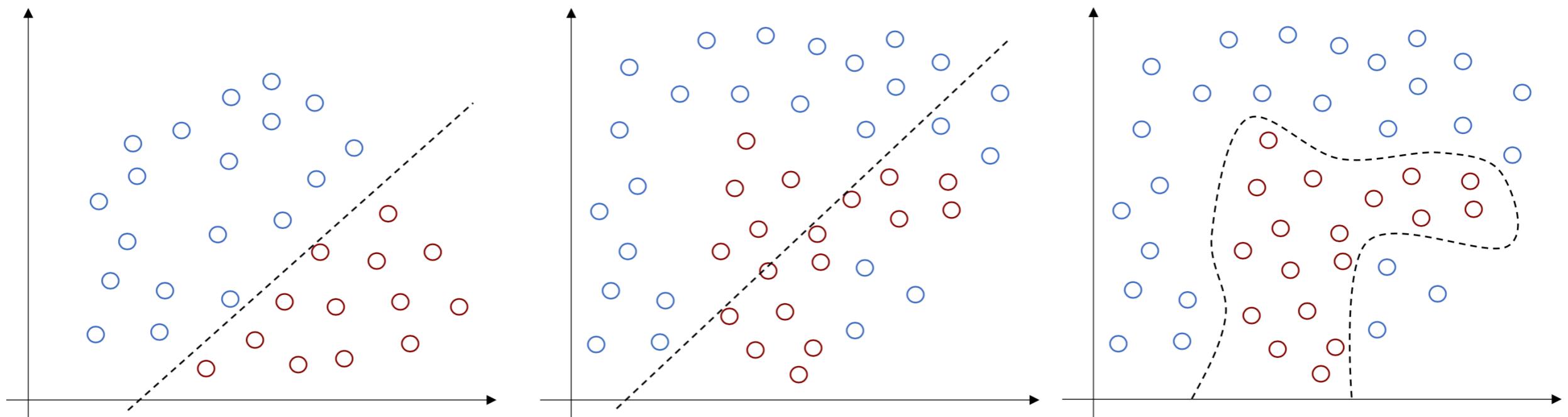
Juntando neuronas



# Red Neuronal Profunda

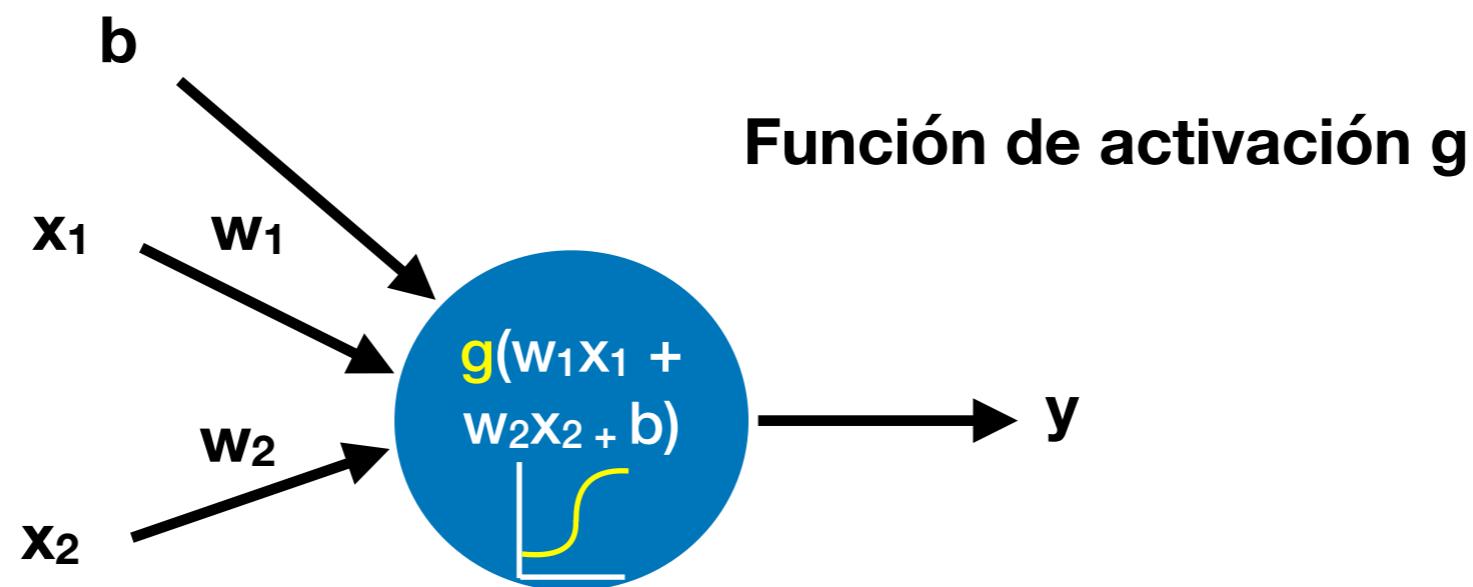
## Función de activación

Los problemas de la vida real a resolver no son lineales, por lo tanto, necesitamos introducir no linearidades a nuestro modelo.



# Red Neuronal Profunda

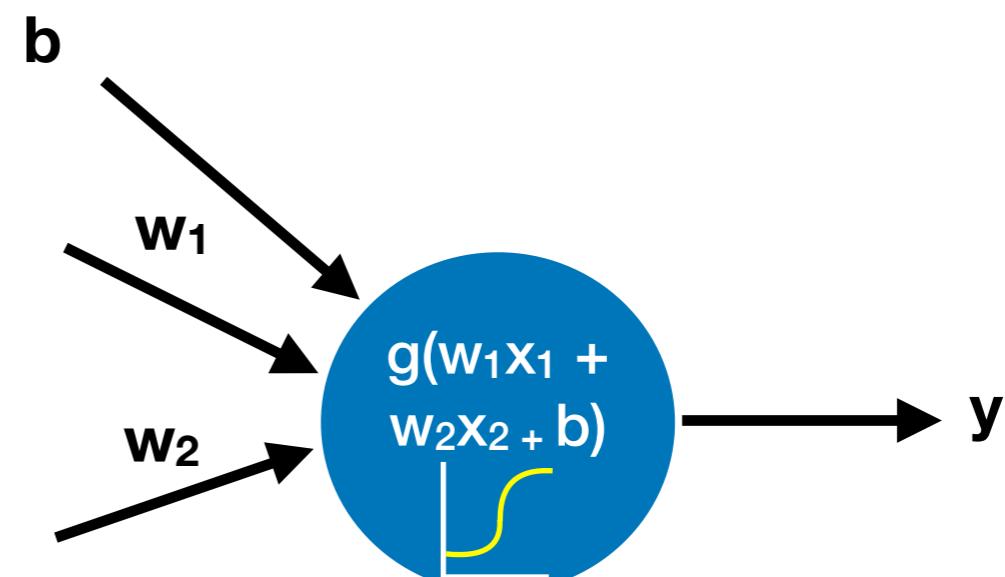
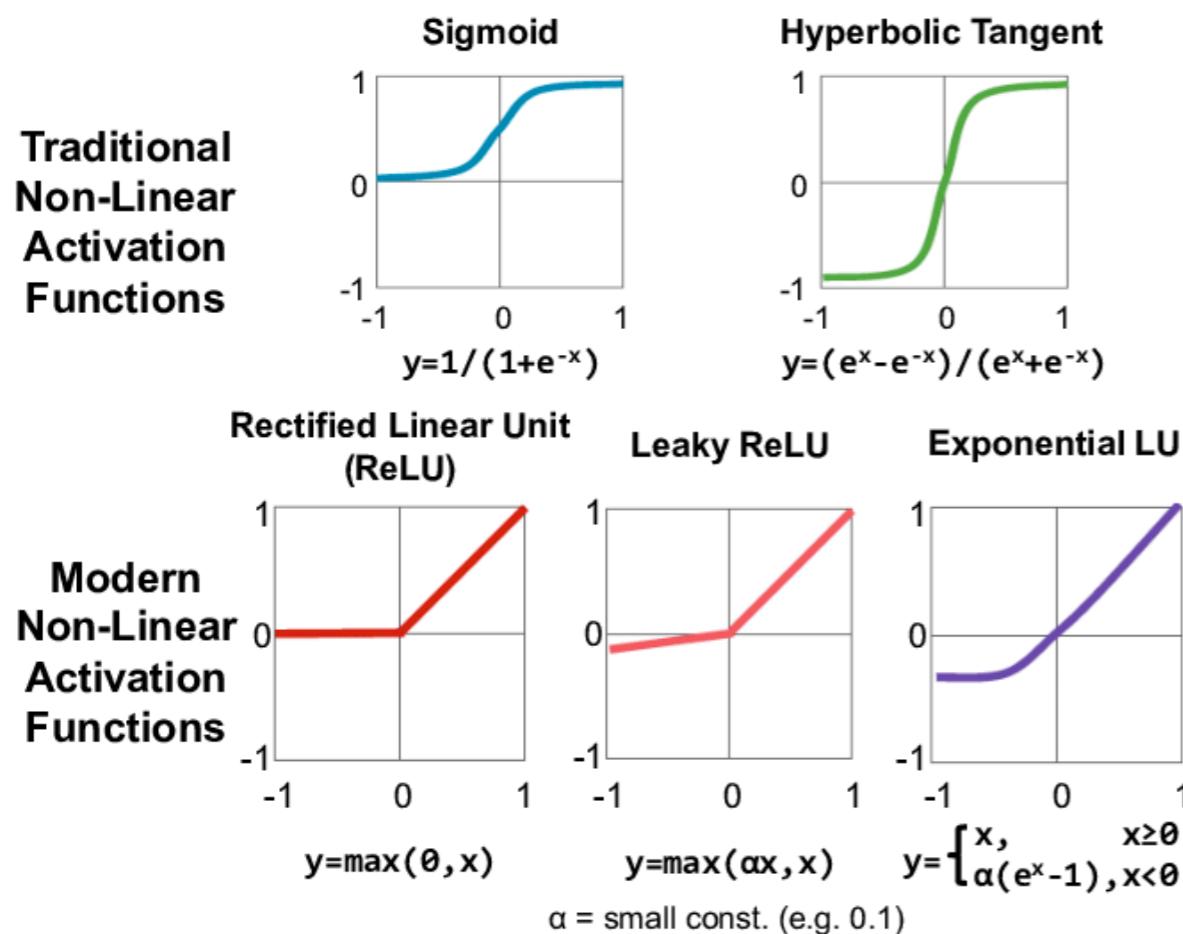
## Función de activación



# Red Neuronal Profunda

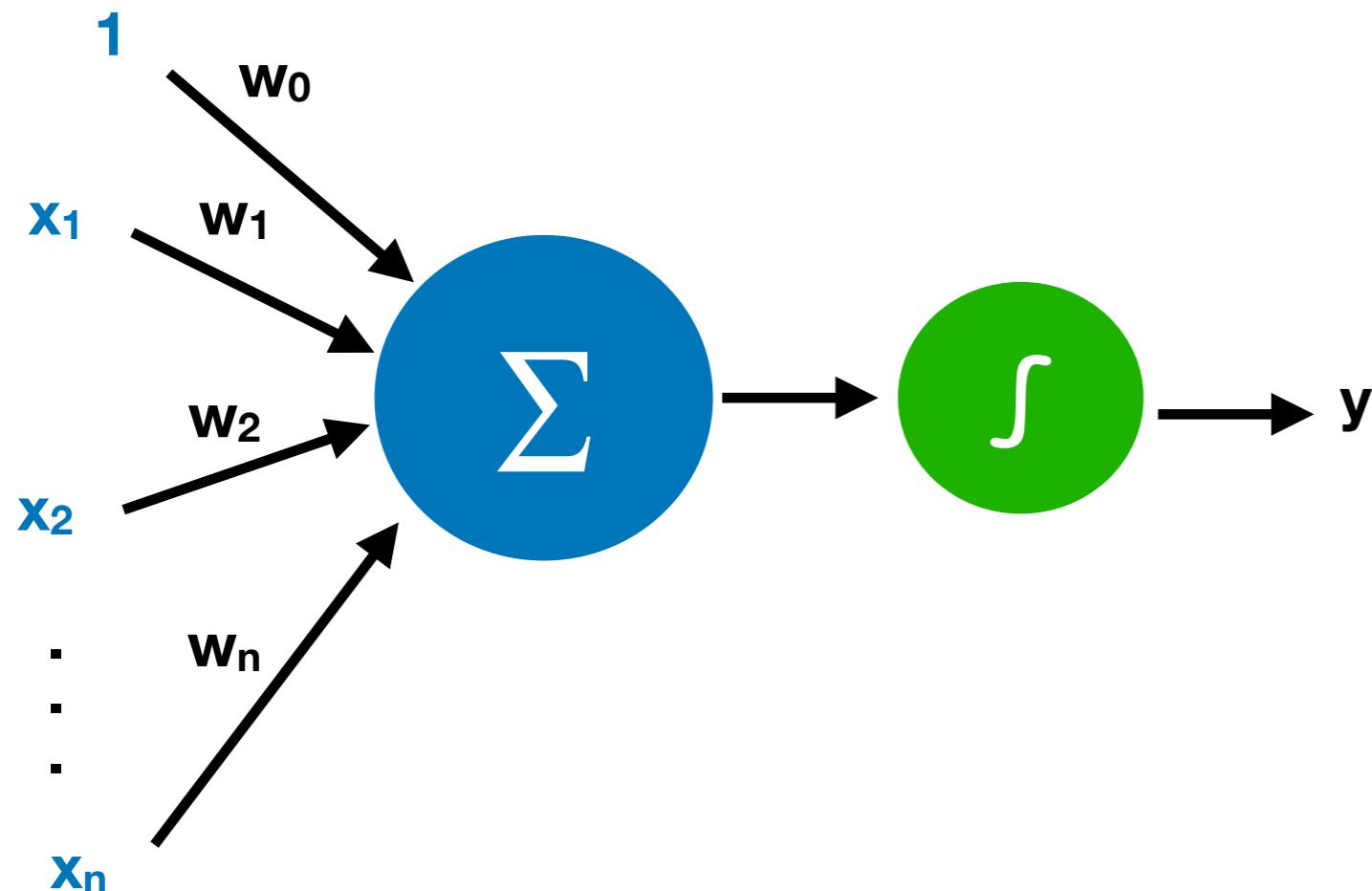
## Función de activación

### Función de activación $f$



# Red Neuronal Profunda

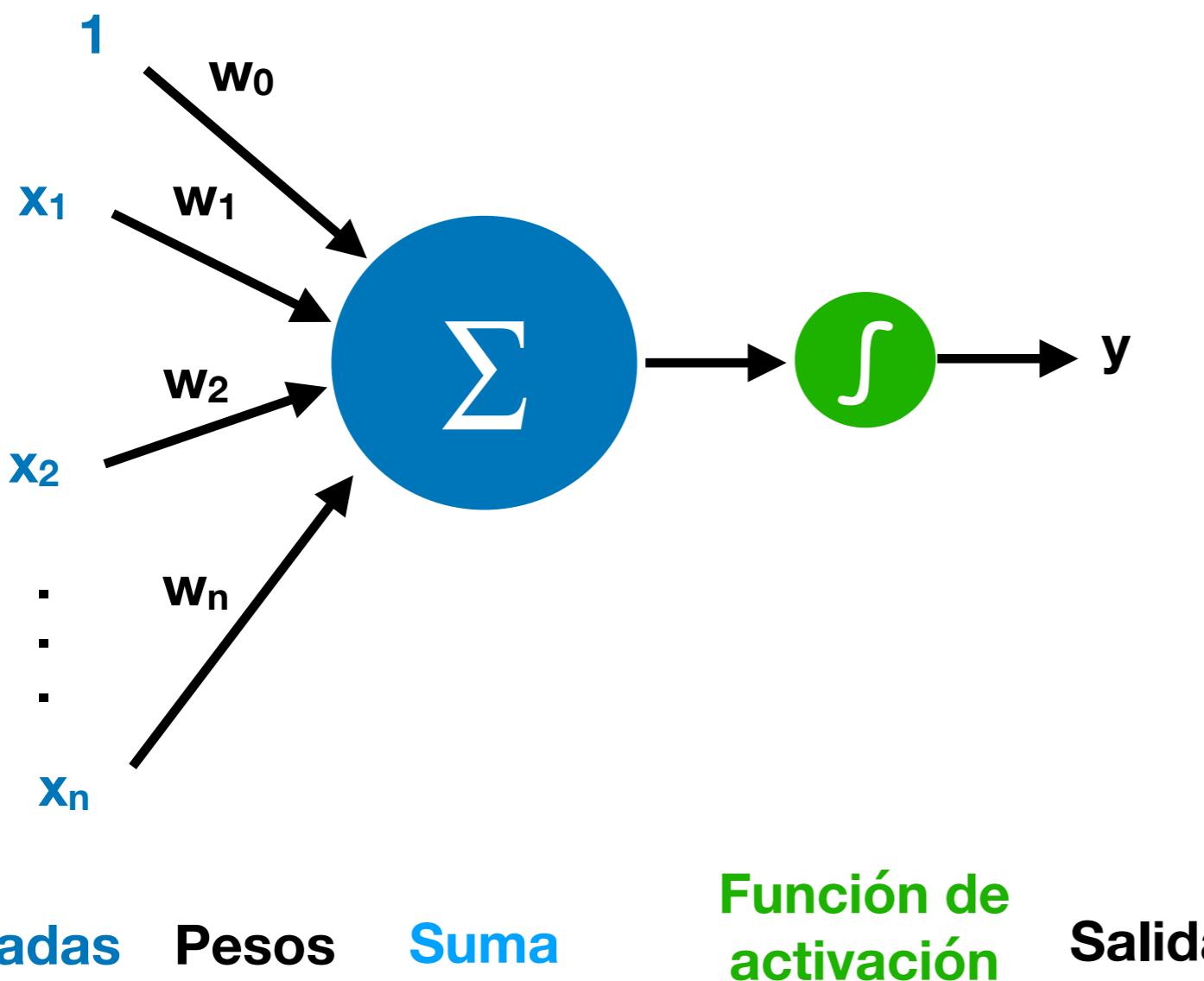
Perceptrón en general



Función de  
activación

# Red Neuronal Profunda

Perceptrón en general

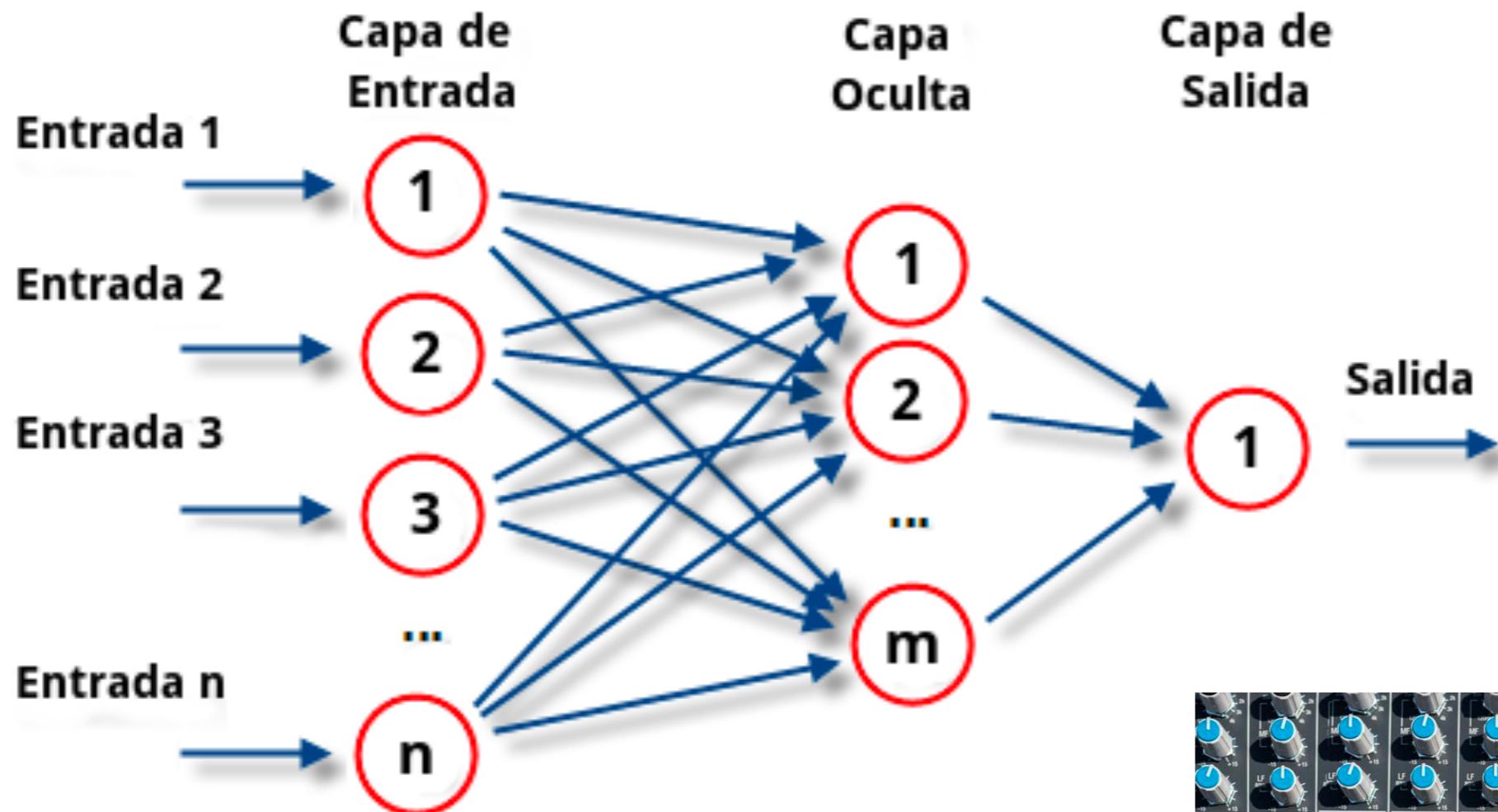


$$y = g(w_0 + \sum_{i=1}^n x_i w_i)$$

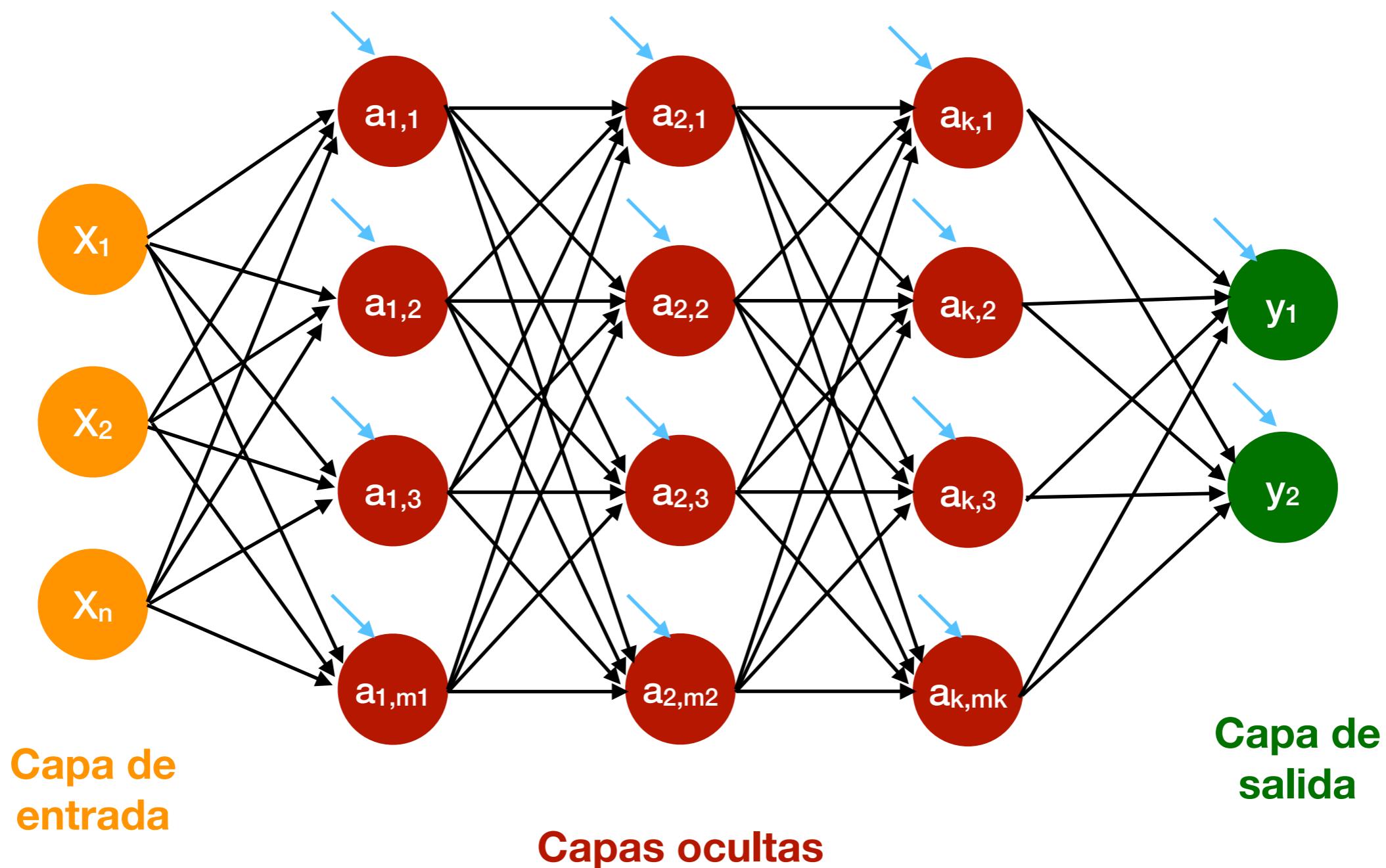
$$y = g(w_0 + X^T W)$$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

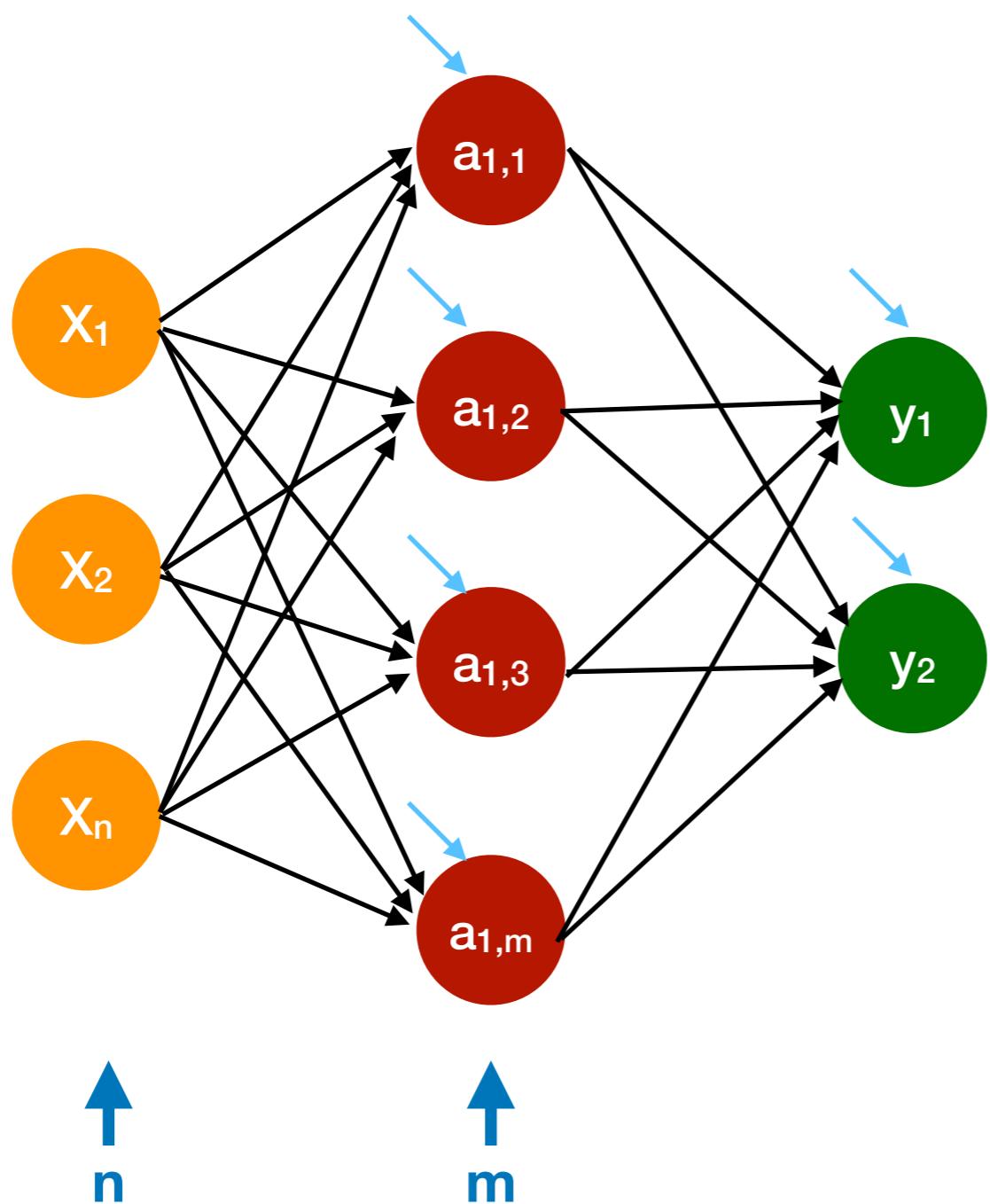
# Red Neuronal Profunda



# Red Neuronal Profunda

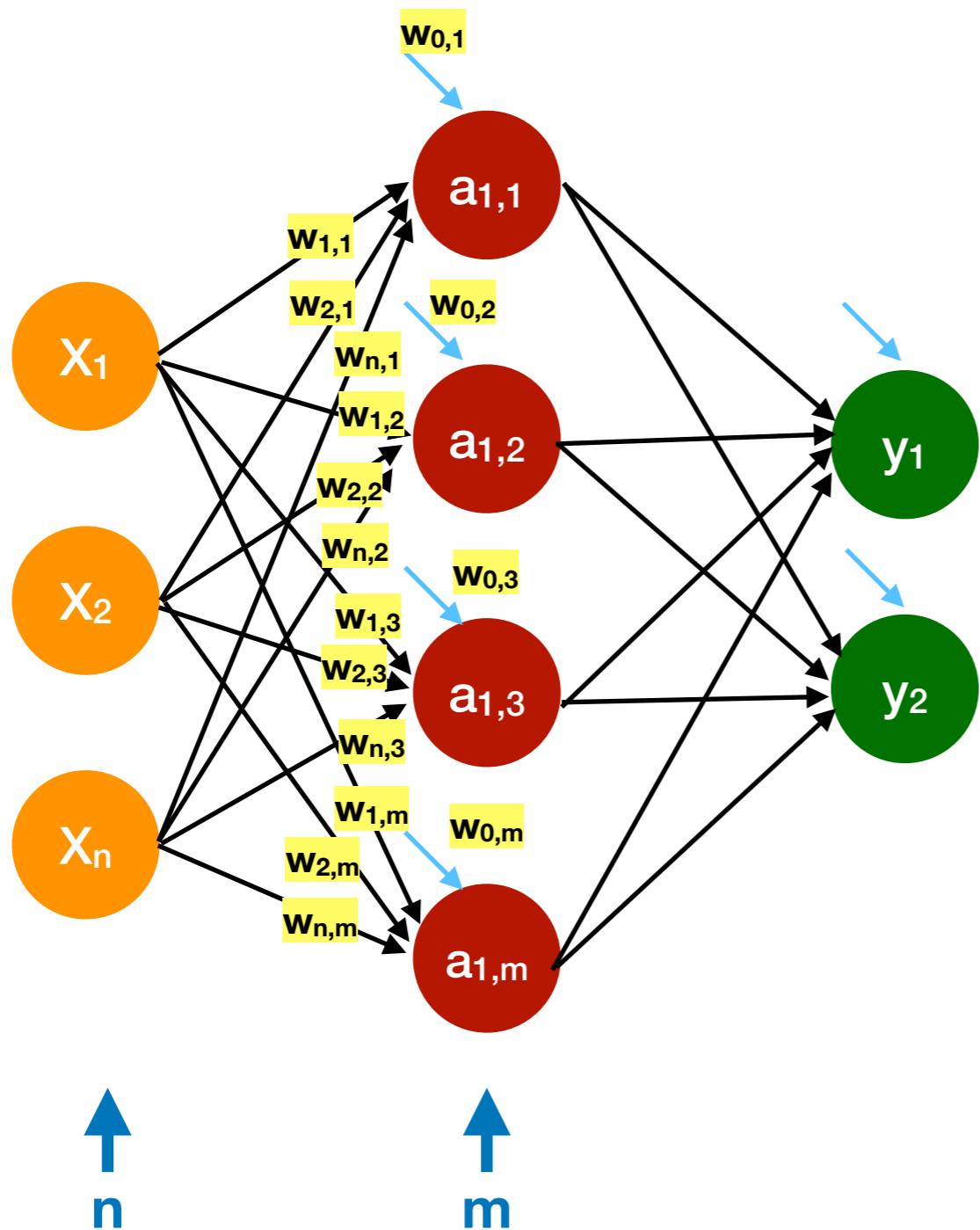


# Red Neuronal Profunda



Número de parámetros a ajustar en la capa oculta:  
 $(n + 1) * m$

# Red Neuronal Profunda



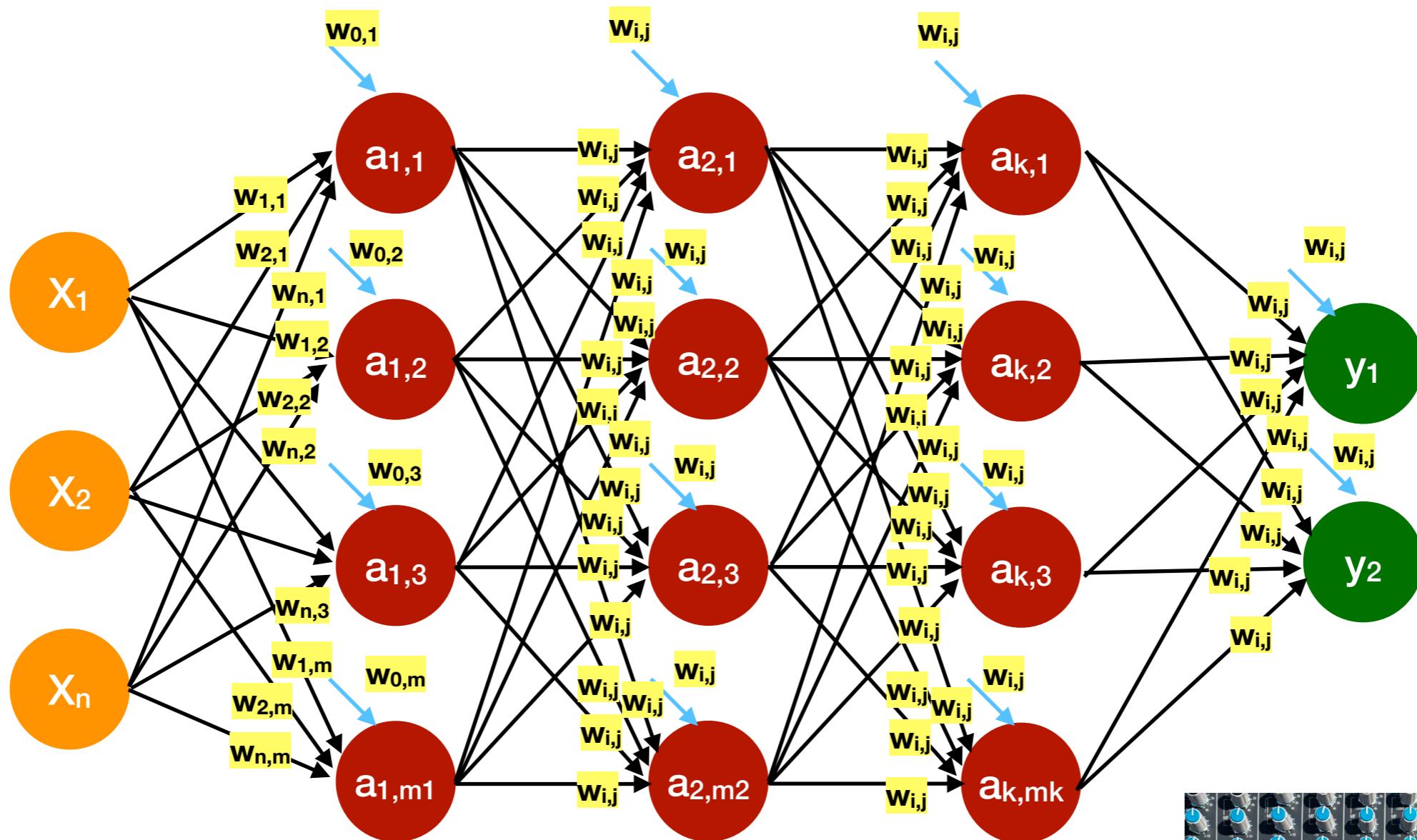
Número de parámetros a ajustar en una capa oculta:

$$(n + 1) * m$$

Siendo  $n$  el n° de nodos en la capa anterior y  $m$  el n° de nodos de la capa a considerar

En el ejemplo:  $(3 + 1) * 4 = 16$

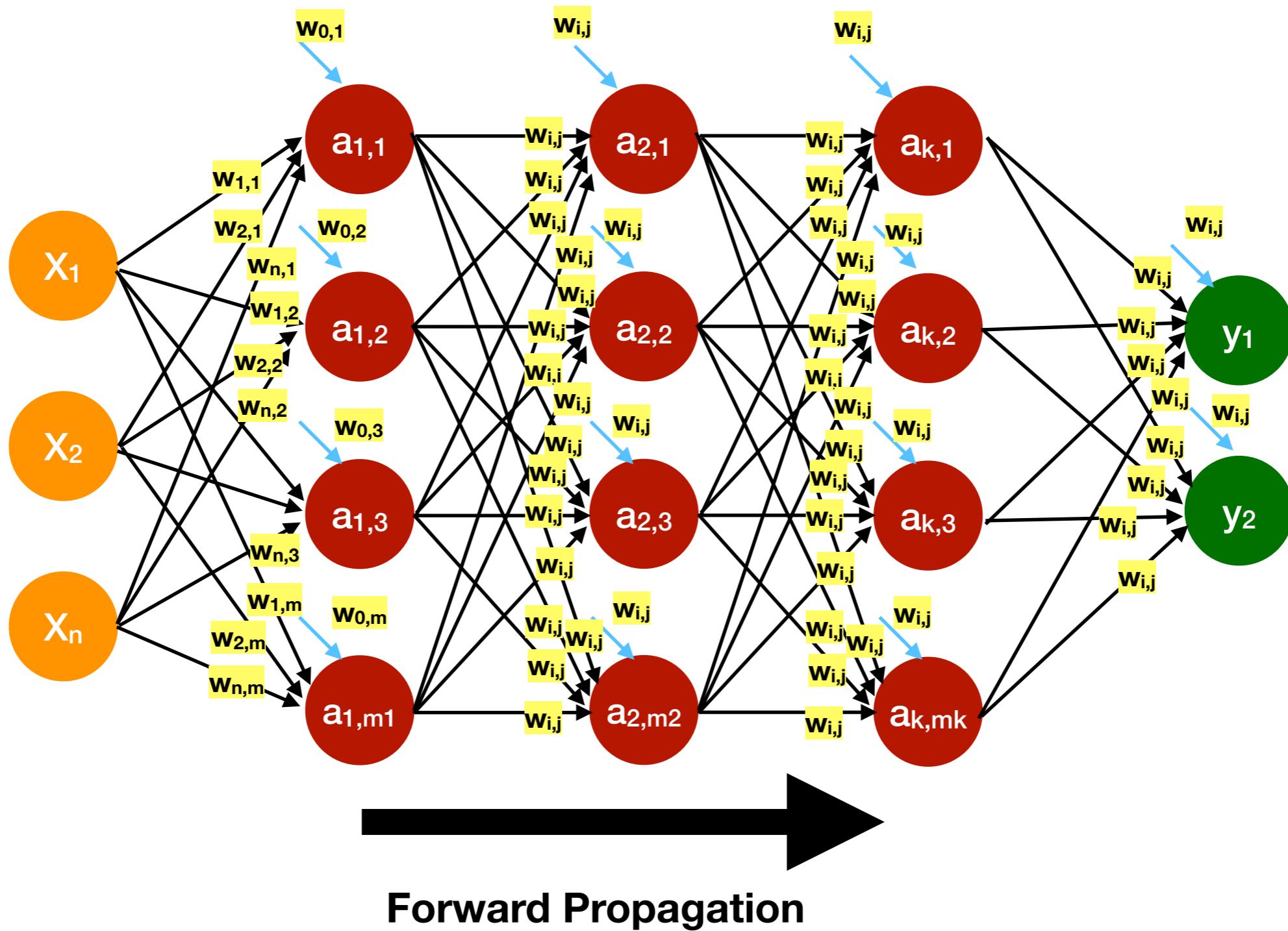
# Red Neuronal Profunda



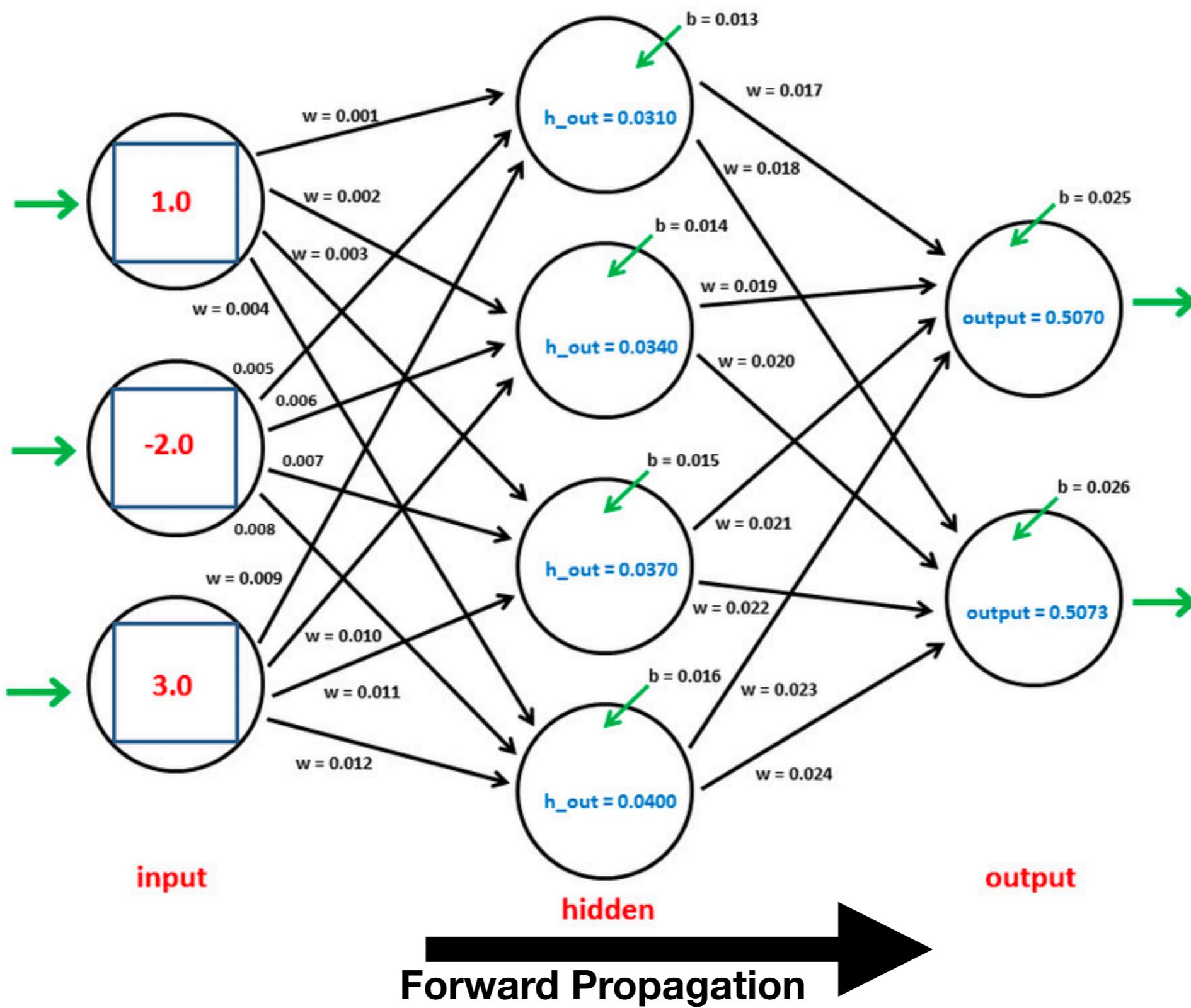
Son todos estos pesos y sesgos los que la red neuronal tiene que ajustar



# Red Neuronal Profunda

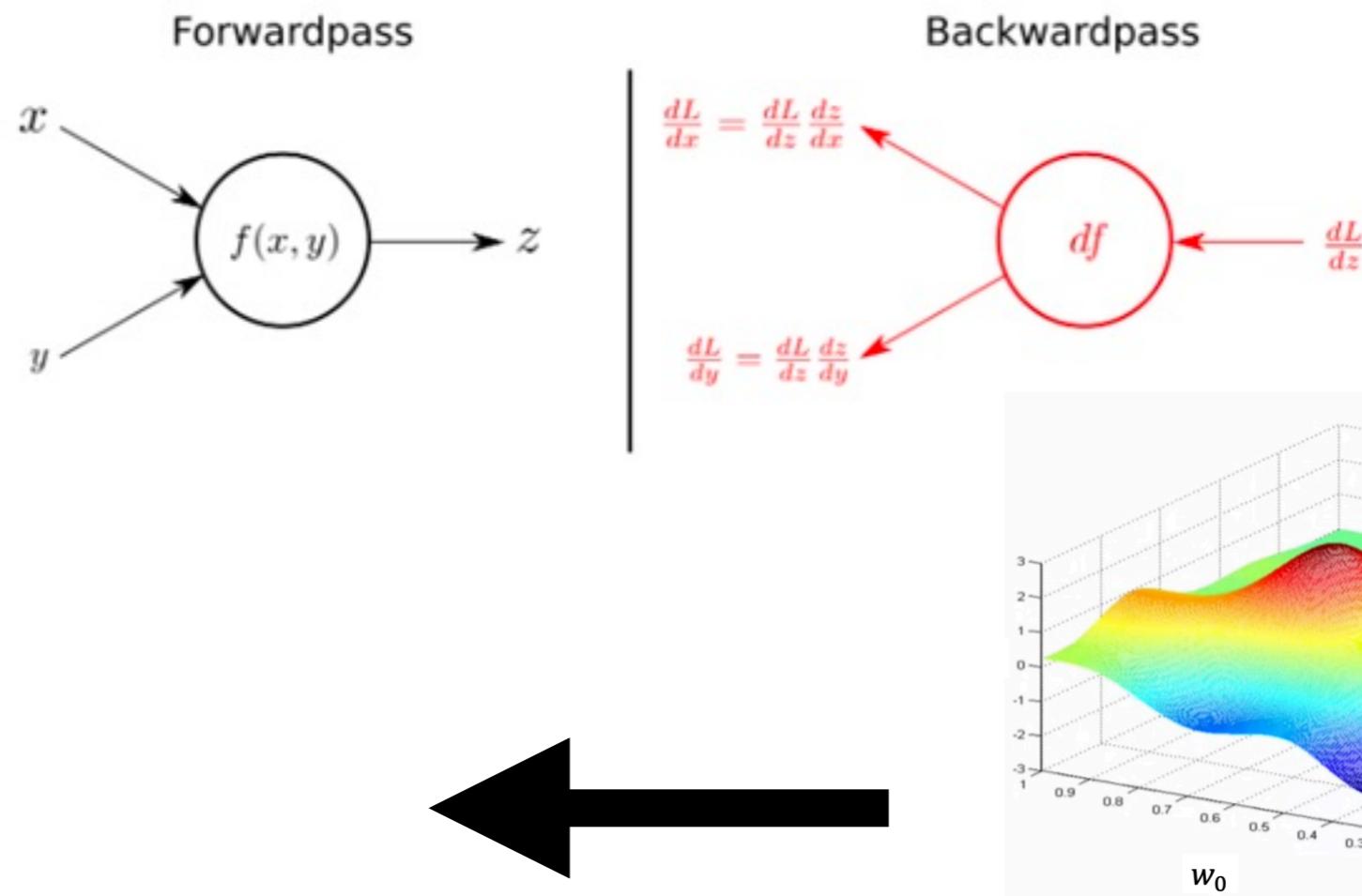


# Red Neuronal Profunda



# Red Neuronal Profunda

Después de una forward propagation se compara el resultado calculado con el real.

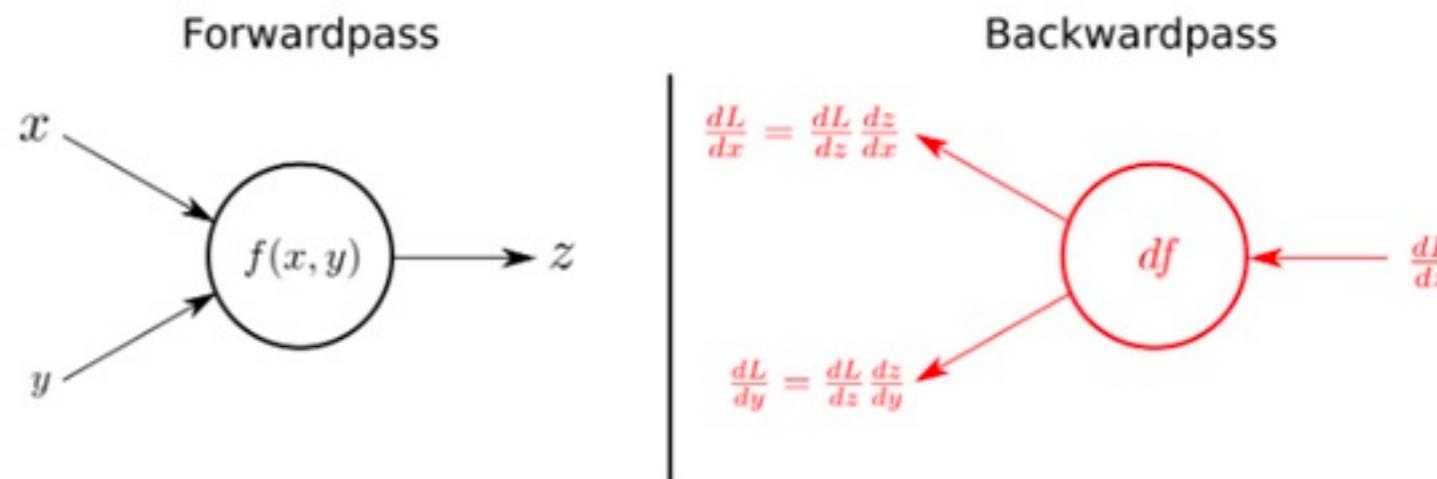


Backwardpropagation  
(Descenso del gradiente)

# Red Neuronal Profunda



Backwardpropagation  
(**Descenso del gradiente**)



## Tasa de aprendizaje (visto en Descenso del Gradiente)

La tasa de aprendizaje define la **magnitud** del cambio que sufren los pesos.

Resulta complicado detectar la tasa de aprendizaje óptima.

Una tasa de aprendizaje demasiado alta sobrepasa el mínimo sin llegar a converger o incluso hace que la red diverja (aumenten las pérdidas).

# Red Neuronal Profunda



**Backward propagation  
(Descenso del gradiente)**

## Ajuste de la Tasa de aprendizaje

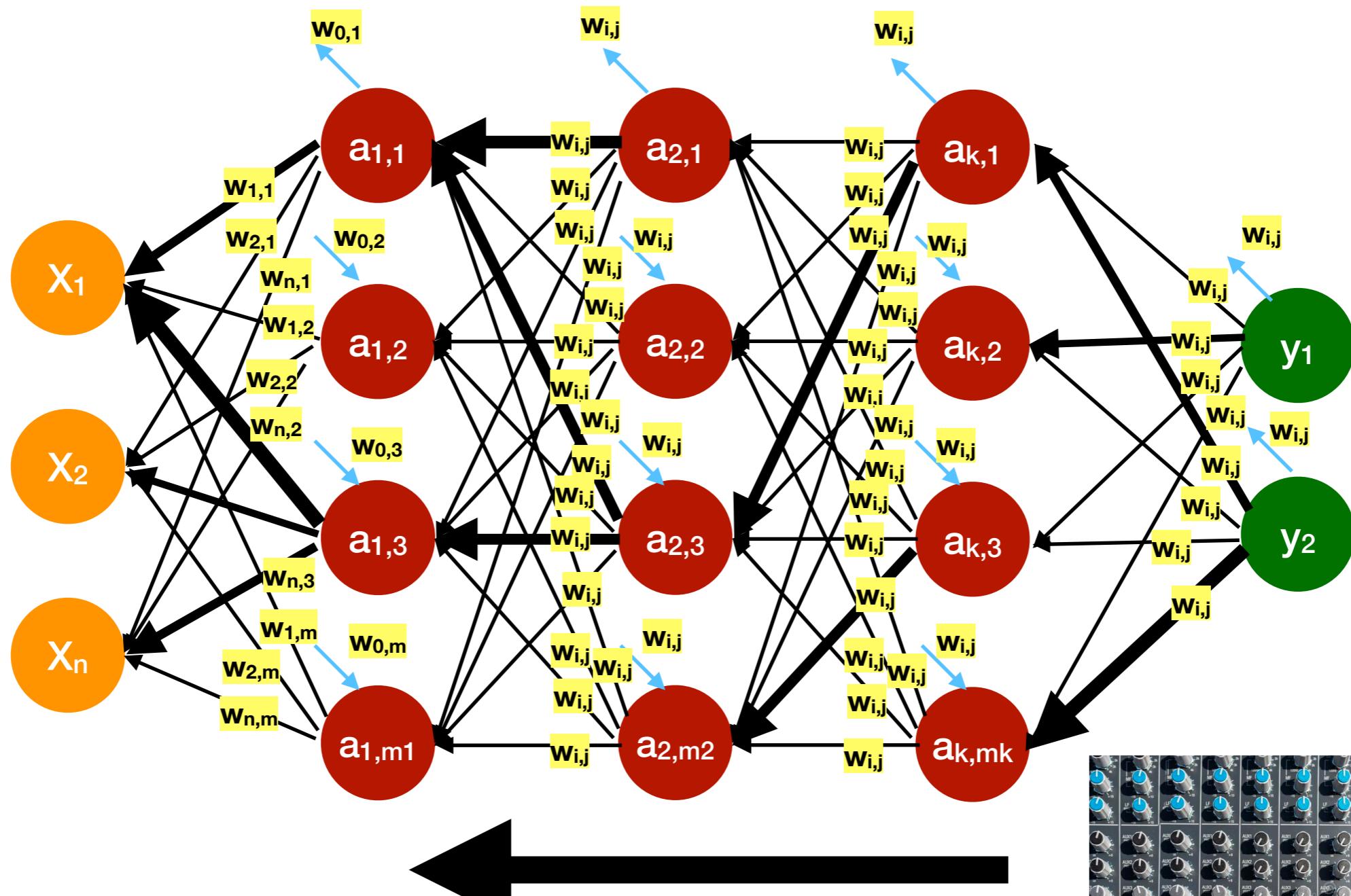
Opción 1: Tasa de aprendizaje constante. Probamos diferentes tasas hasta encontrar la óptima

Opción 2: Optimizadores (Adagrad, Adadelta, RMSprop, Adam). Tasa de aprendizaje adaptativa que aumenta o disminuye dependiendo de como de grande es el gradiente, cómo de rápido se está produciendo el aprendizaje, la magnitud de cada peso particular ... Se emplea una tasa de aprendizaje para cada peso.

```
from tensorflow.keras.optimizers import Adagrad, Adadelta, RMSprop, Adam

optimizer_adagrad = Adagrad(lr=0.01, epsilon=1e-08, decay=0.0)
optimizer_adadelta = Adadelta(lr=1.0, rho=0.95, epsilon=1e-08, decay=0.0)
optimizer_rmsprop = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
optimizer_adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
```

# Red Neuronal Profunda



Cálculo  
del error  
cometido

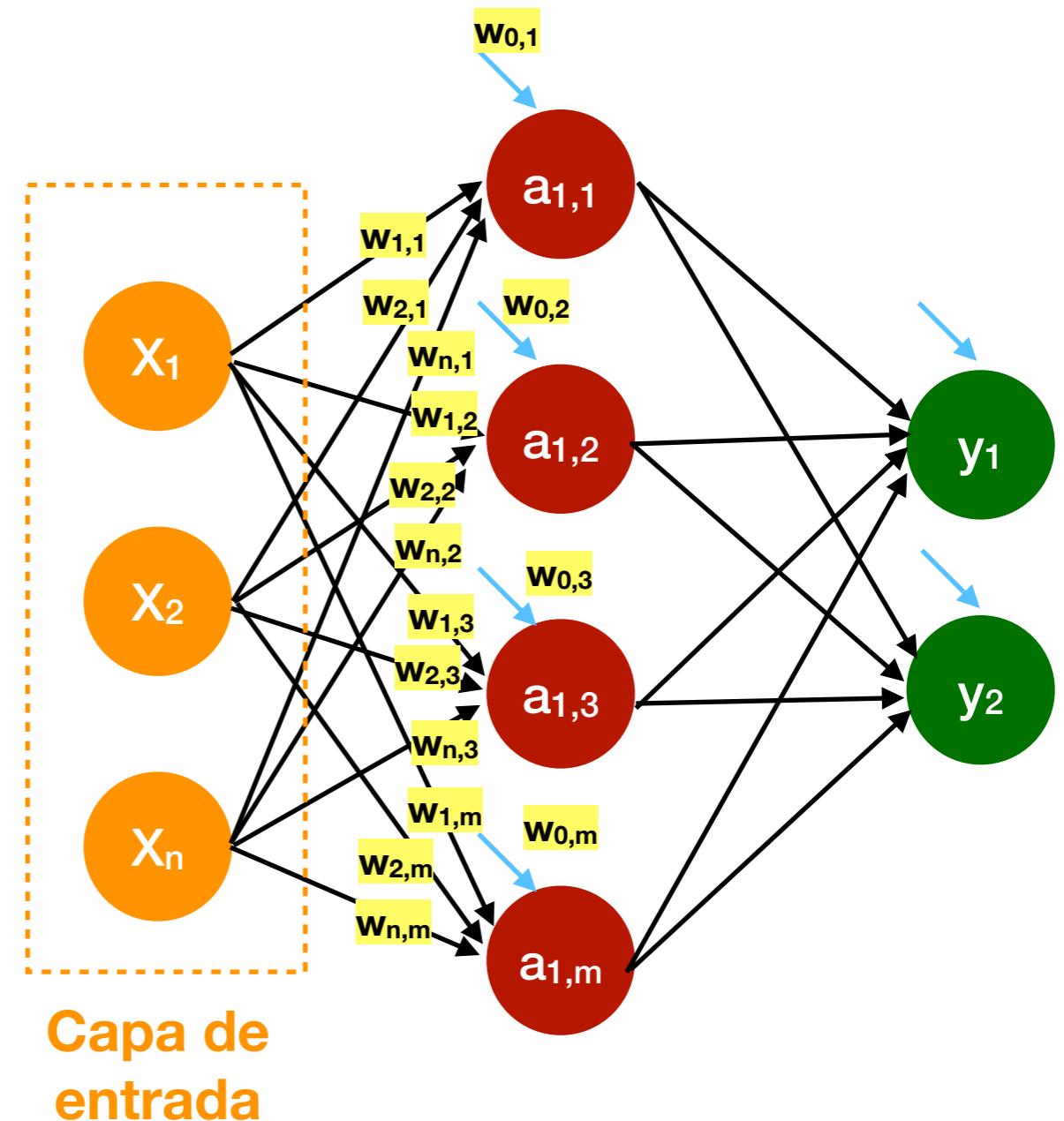


# Red Neuronal Profunda

## Capa de entrada:

Tiene tantos nodos como variables de entrada tenga:

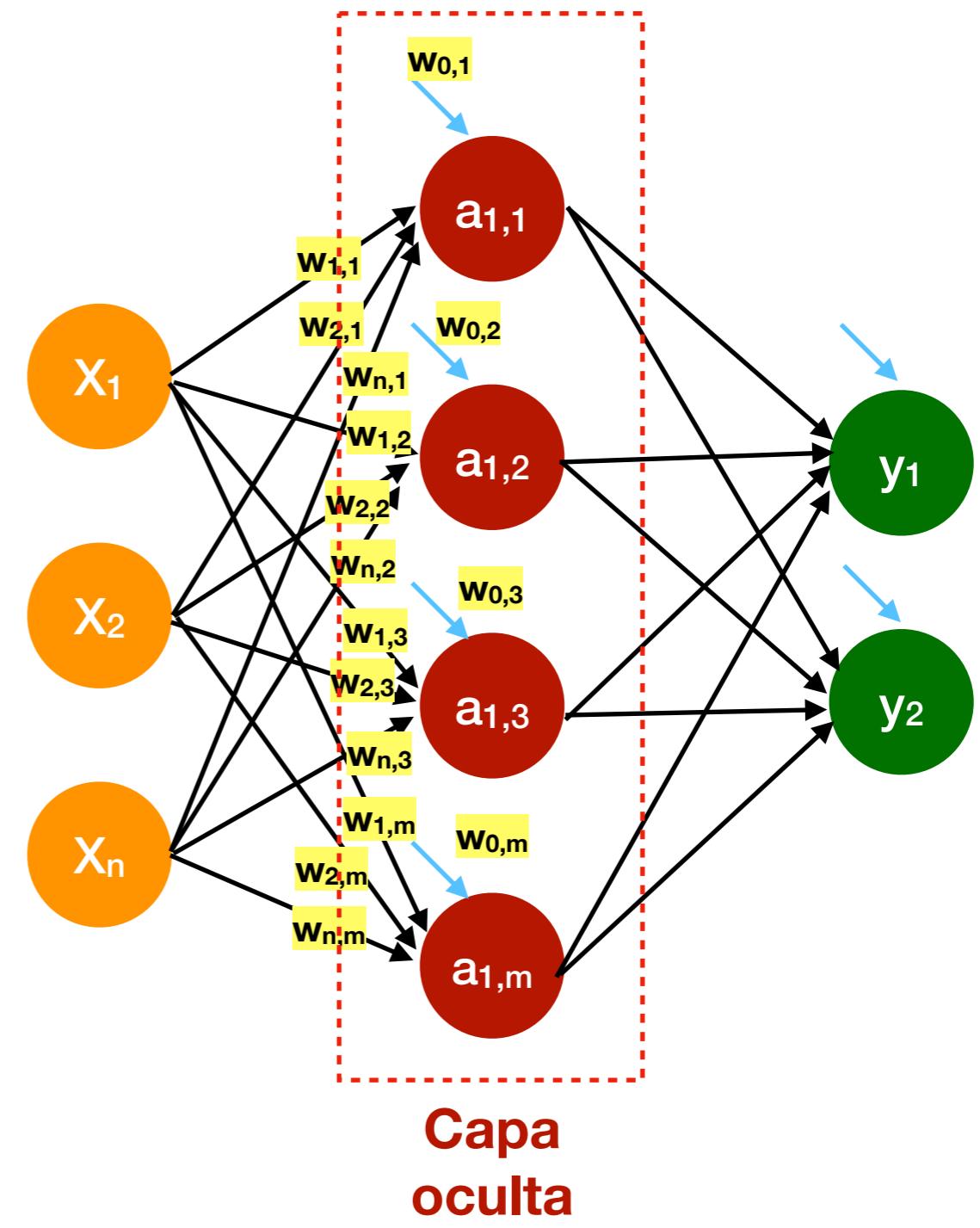
- Características de una vivienda.
- Histórico de temperaturas.
- Estilo de vida de una persona.
- ...



# Red Neuronal Profunda

## Capa oculta:

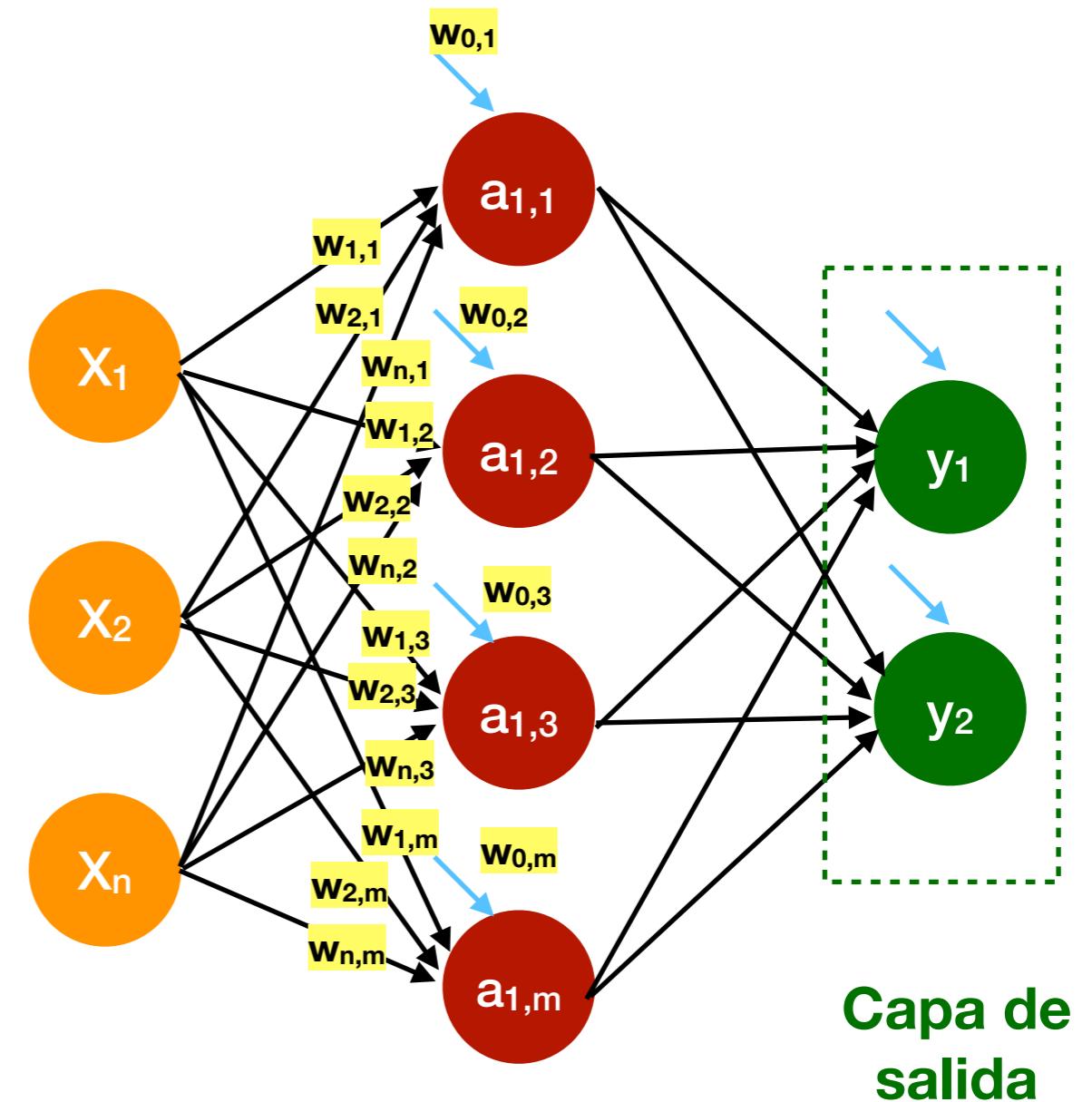
- Una red neuronal puede estar formada por **múltiples capas ocultas con múltiples neuronas**.
- Son todas aquellas capas entre la entrada y la salida de la red
- **Entrada:** variables de entrada/activaciones de la capa anterior, ponderadas por los pesos de las conexiones.
- **Salida:** activaciones de las neuronas de la capa
- A la hora de diseñar la arquitectura el programador debe definir el número de capas ocultas, el número de neuronas por capa y la función de activación de cada capa (**hiperparámetros**).



# Red Neuronal Profunda

## Capa de salida:

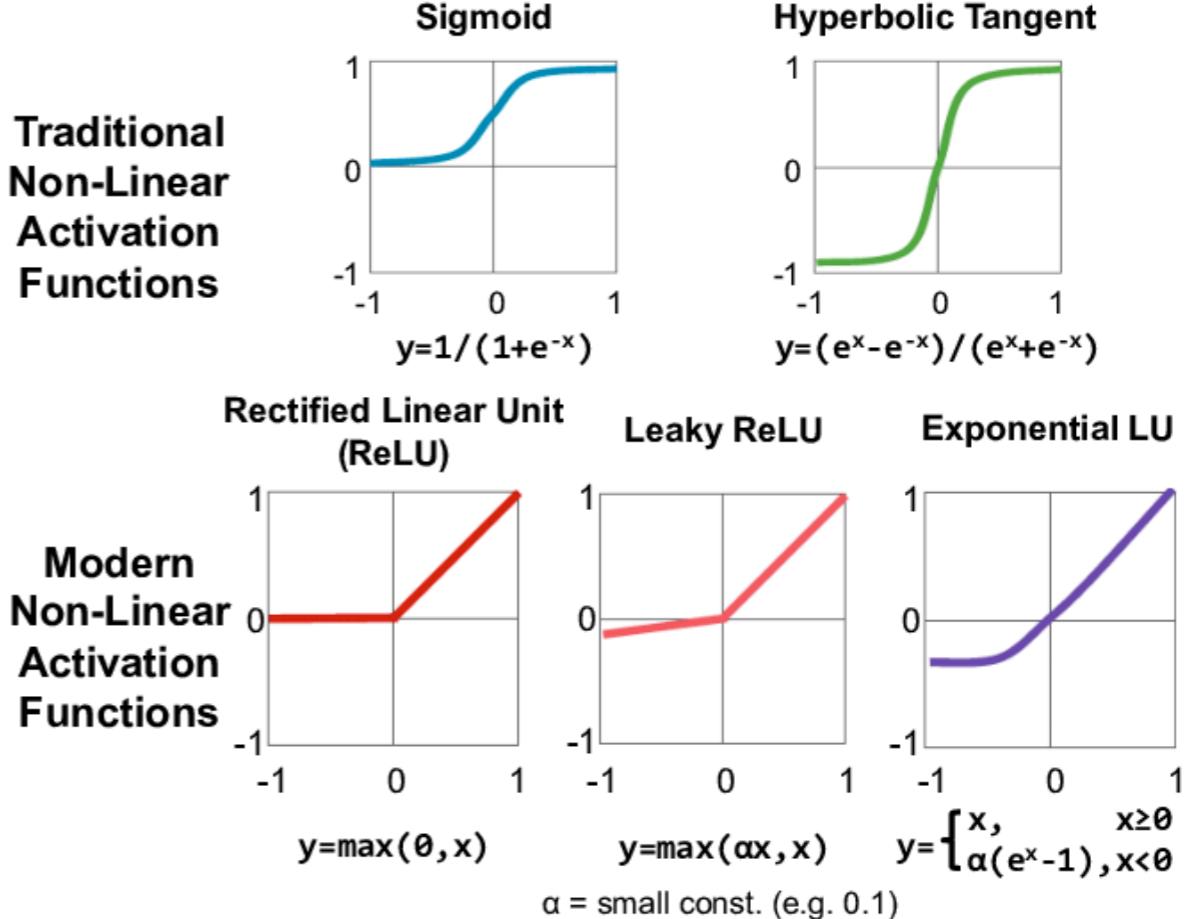
- Nos va a dar la salida (**predicción**) de la red neuronal.
- El **número de neuronas** está directamente ligado con el problema a resolver.
- La **función de activación** dependerá del tipo de problema a resolver (clasificación binaria, clasificación multi-clase, regresión)



# Red Neuronal Profunda

Recomendaciones a la hora de seleccionar la **función de activación**:

- **Sigmoide** y **tanh** no son las más eficientes debido al problema del desvanecimiento del gradiente.
- **ReLU** es la elección más recomendada para las capas ocultas. Si sufrimos el problema de “dying relu” podemos utilizar alguna modificación como **leaky ReLU**, **ELU**, etc.
- Para la capa de salida:
  - Si estamos trabajando con regresión, utilizar activación **lineal**.
  - Si estamos trabajando con clasificación, utilizar activación **sigmoide** (binaria) o **softmax** (multiclas).



# Red Neuronal Profunda

Ejemplo implementación:

- 16 valores de entrada.
- 1 capa oculta con 8 neuronas.
- 1 capa de salida con 2 posibles salidas.



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

model = Sequential()
model.add(Input(shape=(16,)))
model.add(Dense(8, activation="relu"))
model.add(Dense(2, activation="softmax"))
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense

input_layer = Input(shape=(16,))
hidden_layer = Dense(8, activation="relu")(input_layer)
output_layer = Dense(2, activation="softmax")(hidden_layer)

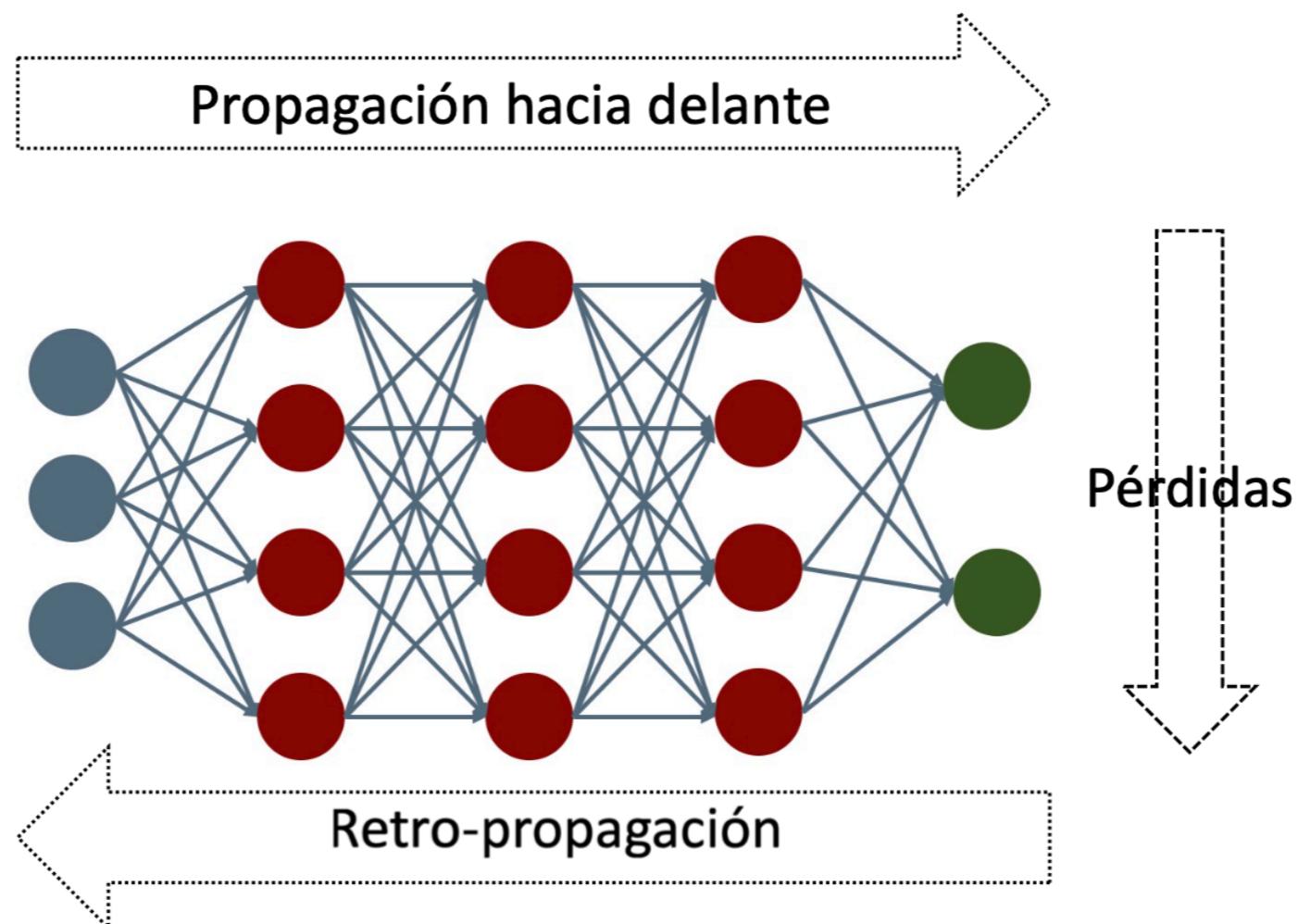
model = Model(inputs=input_layer, outputs=output_layer)
```

# Red Neuronal Profunda

## Función de coste o pérdida (Loss function)

Mide el **error medio** producido en todo el conjunto de datos entre el valor predicho por la red neuronal y el valor real.

Durante el proceso de entrenamiento se van **actualizando los pesos** de las conexiones para minimizar la función de coste.

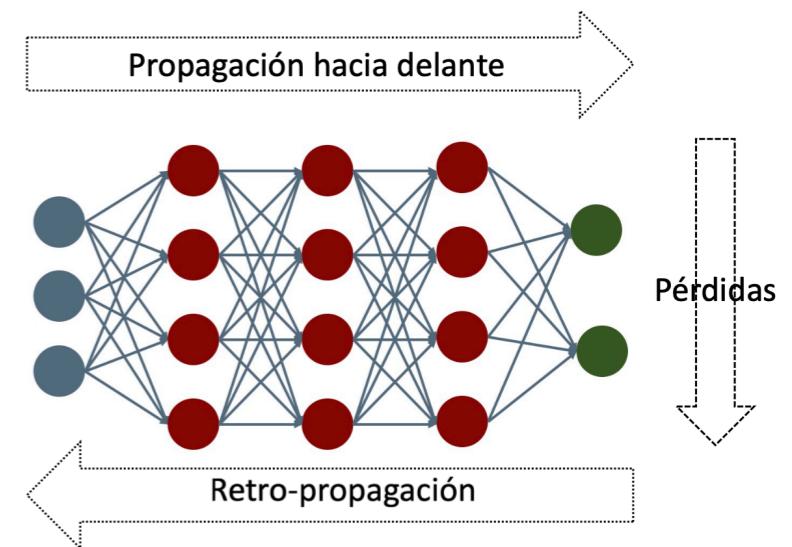


# Red Neuronal Profunda

## Función de coste o pérdida (Loss function)

**Problemas de regresión:** Error cuadrático medio

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



**Problemas de clasificación:** Entropía cruzada → Mide como de lejos la probabilidad predicha se encuentra de la clase real. Aumenta conforme la diferencia entre la probabilidad predicha diverge de la etiqueta real.

$$J(W) = -\frac{1}{n} \sum_{i=1}^n (y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i))$$

Si hay más de dos clases, se calcula la entropía cruzada para cada clase por observación y se suma el resultado.

<https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e>

# Red Neuronal Profunda

## Entrenamiento

- El entrenamiento de una red neuronal es un proceso **iterativo** en el que se realizan predicciones sobre las muestras de entrenamiento y se ajustan los pesos para minimizar la función de coste.
- Cada iteración en la que se produce una predicción y se actualizan los pesos se conoce como **época** (del inglés, *epoch*).
- El número de épocas es un **hiperparámetro** a definir por el desarrollador.

# Jugando con Redes Neuronales

<https://playground.tensorflow.org>