CIS

Introducción a las Redes Neuronales

Machine Learning
Fernando Teshima

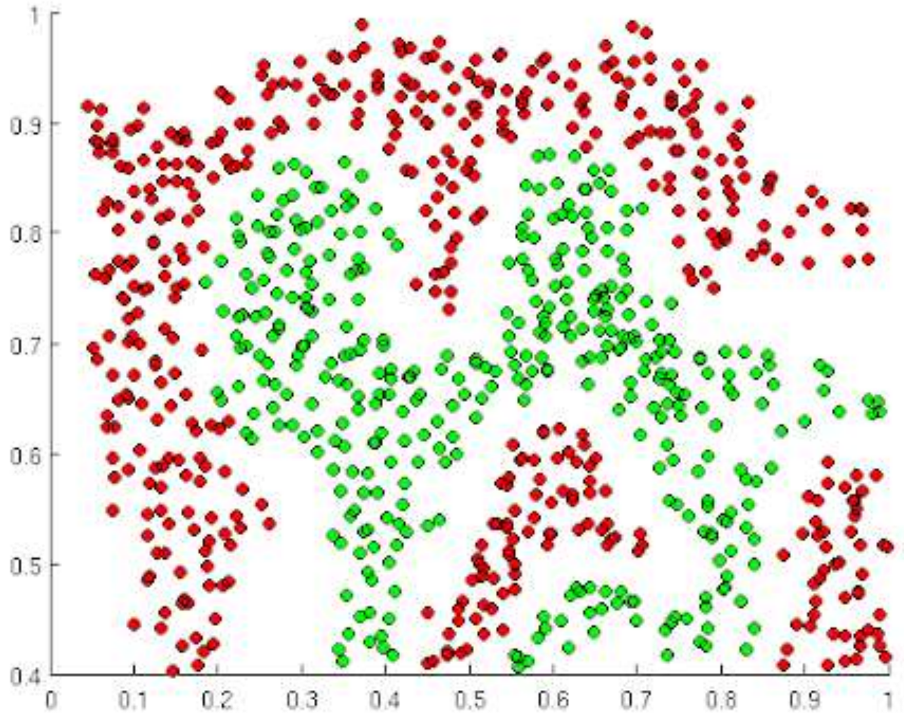
UNIVERSIDAD DE LIMA

Limitantes en la Regresión

- Pueden ser inflexibles
  - Podemos compensar con polinomios, pero hasta un punto
  - Terminamos con demasiadas variables de entrada

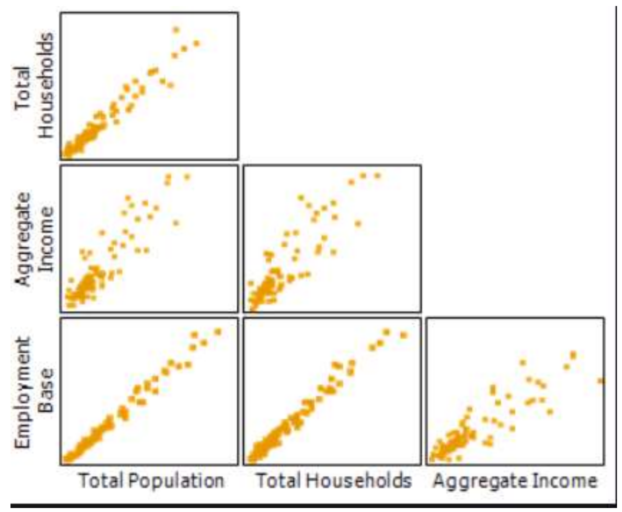
Por ejemplo con para 100 features, si los transformáramos a términos cuadráticos, tenemos ahora 4950 features.

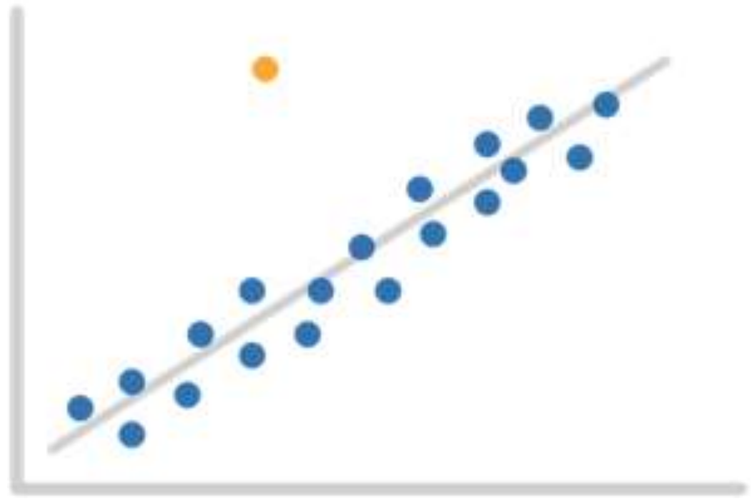
$$\frac{n!}{(n-r)!\,r!} = \binom{100}{2} = 4950$$



Limitantes en la Regresión

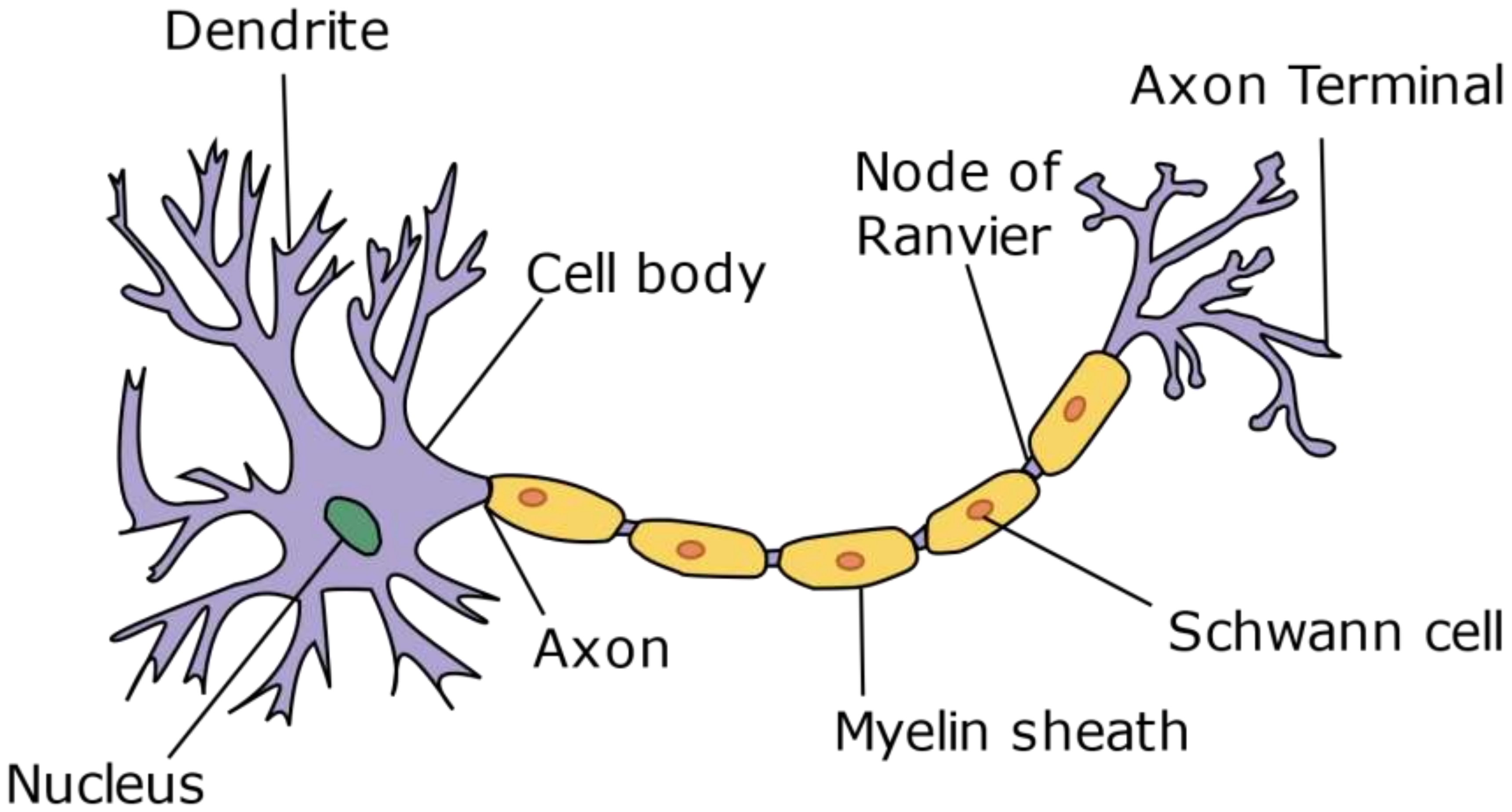
- Asume que las variables independientes son también independientes entre si (no son colineales)
- Outliers





Redes Neuronales

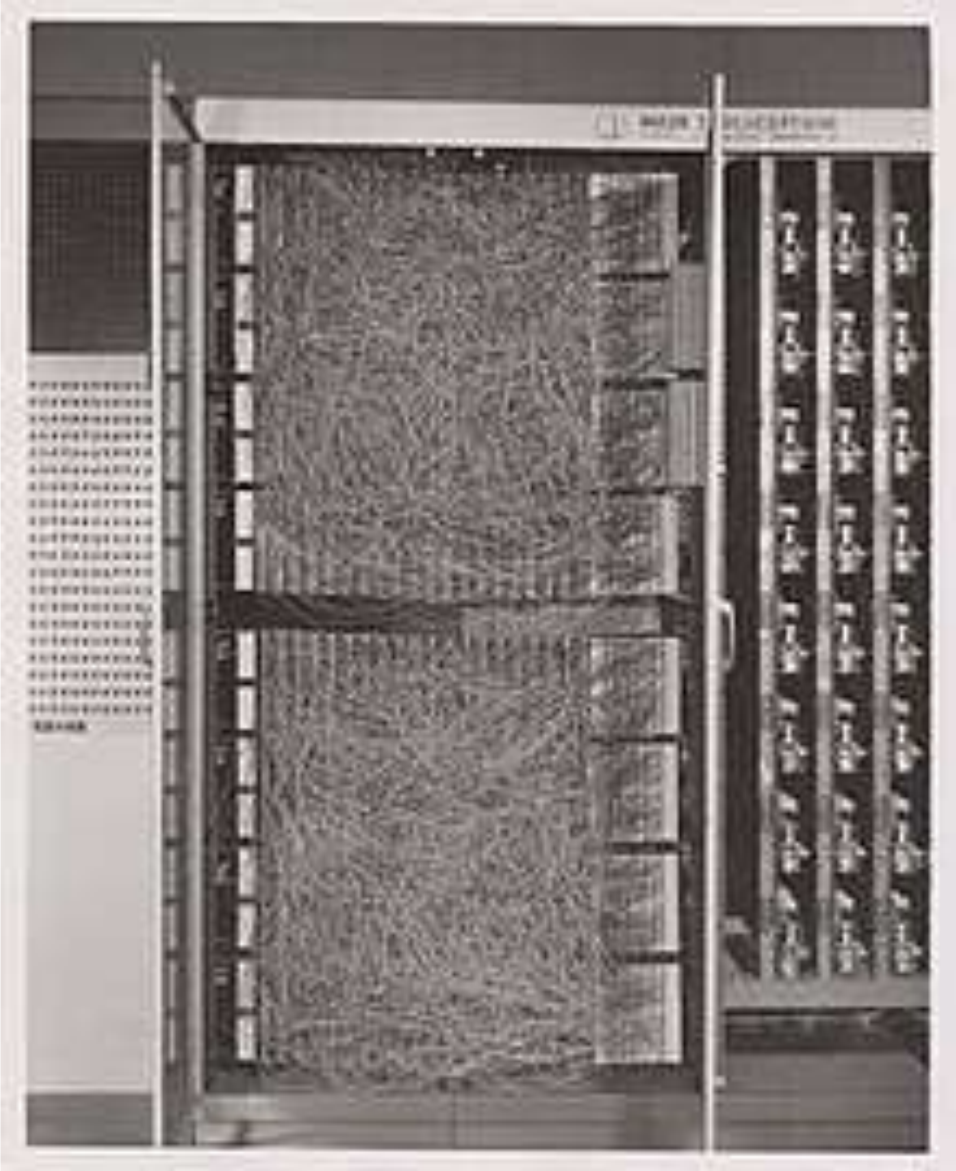
La Neurona

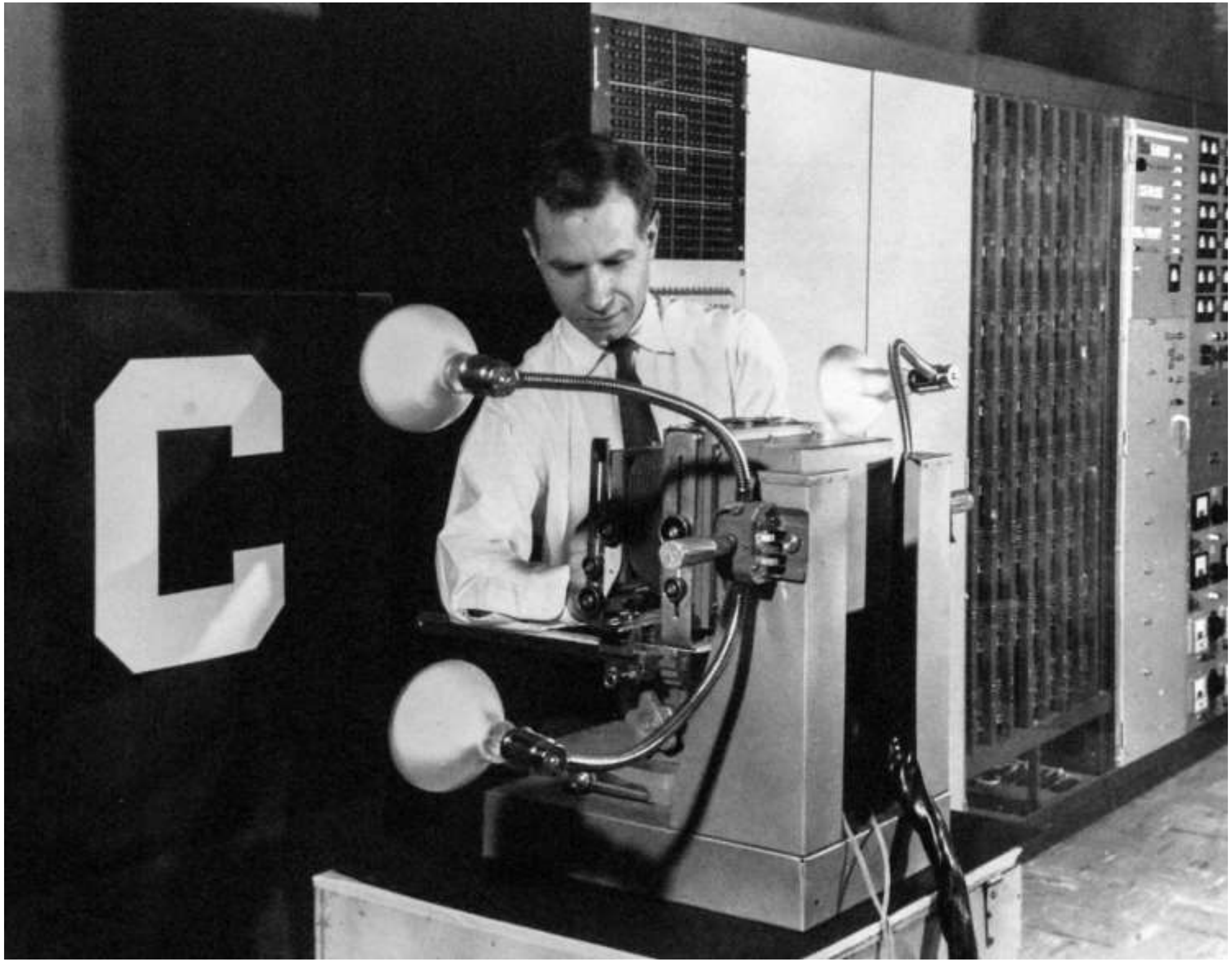


Fuente: https://commons.wikimedia.org/wiki/File:Neuron.svg

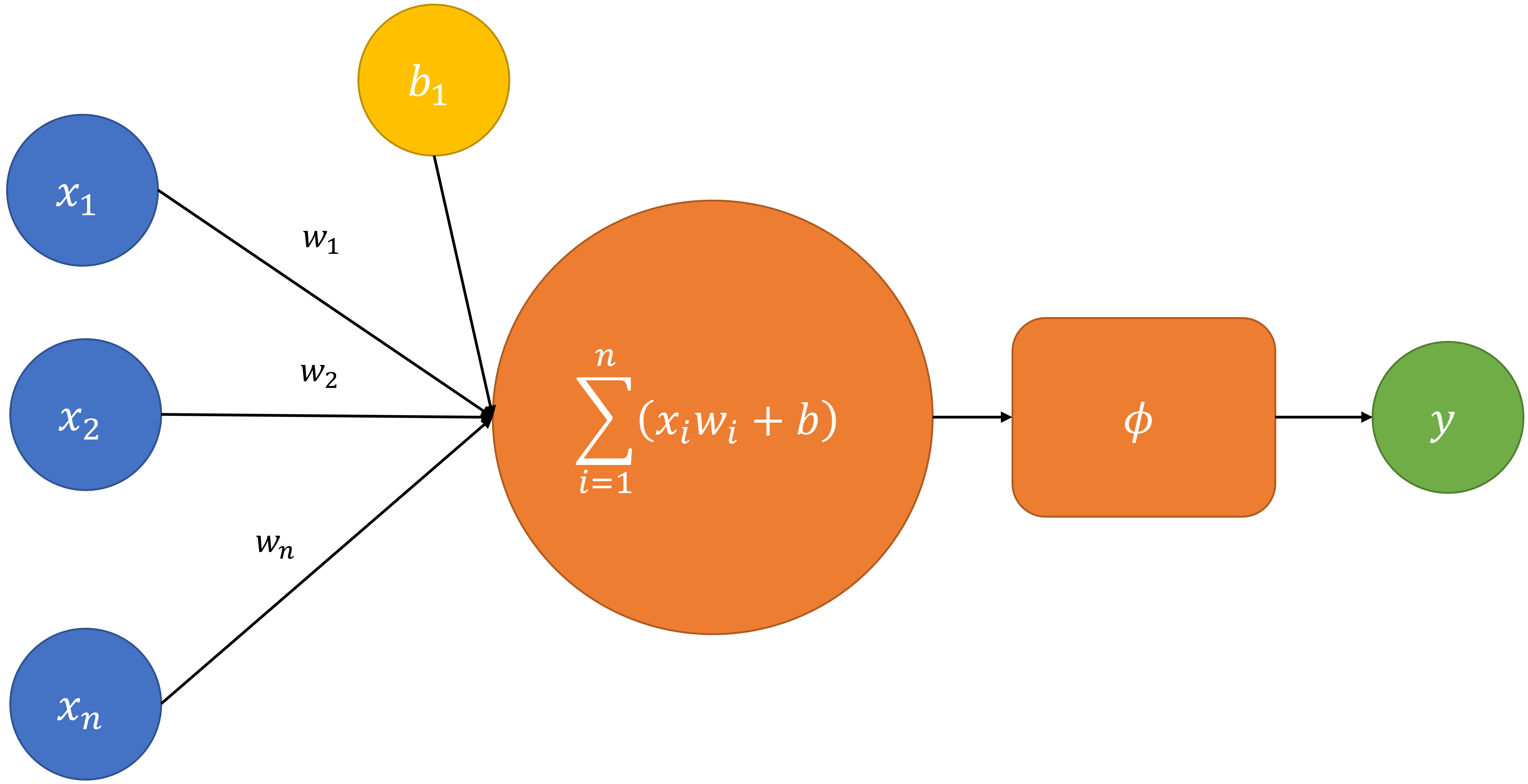
Historia del Perceptrón

- Creado por Frank Rosenblatt en 1957
- Originalmente ideado para ser una máquina
- Reconocimiento de imágenes
- Fotodetectores y potenciómetros y motores



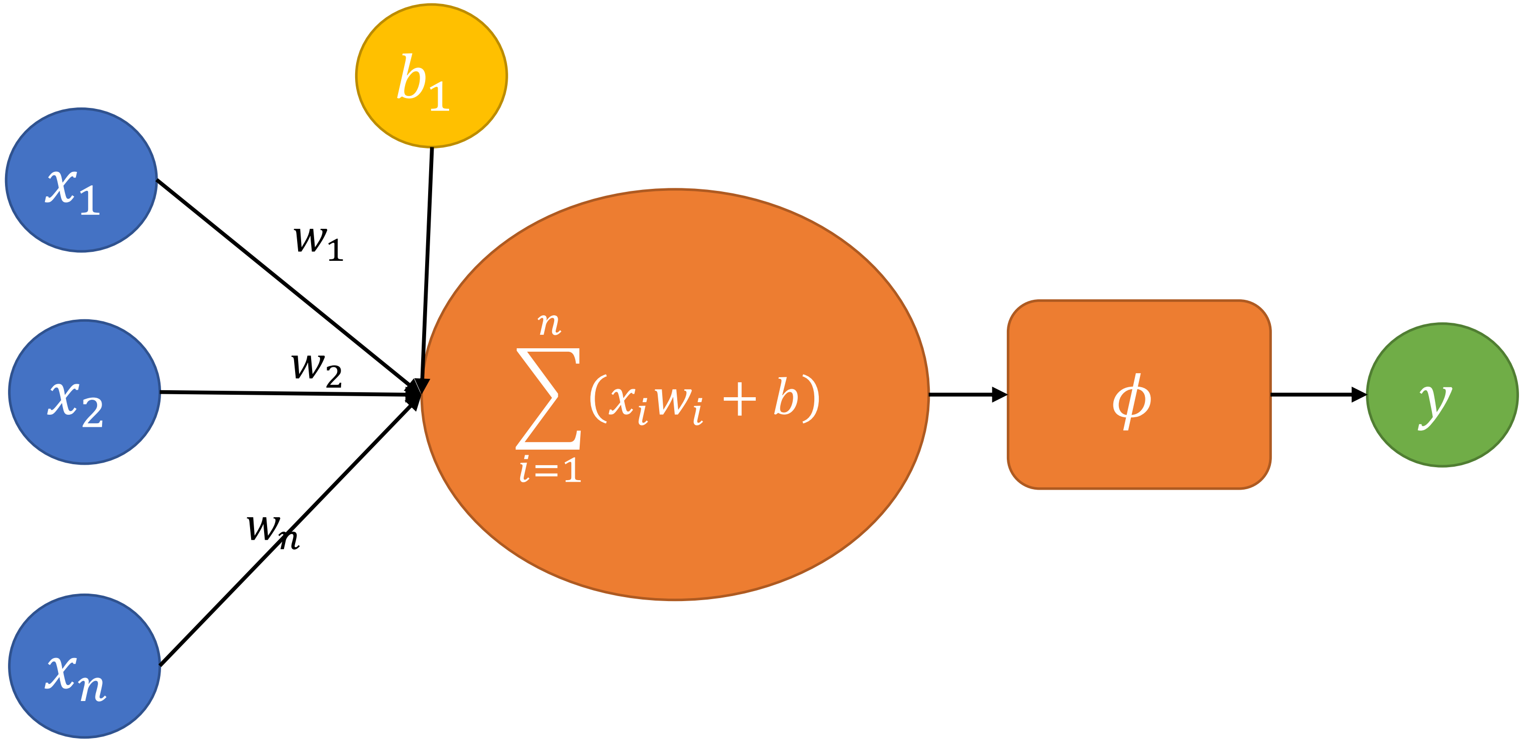

C

Perceptrón



Partes del Perceptrón

1) Entrada
2) Pesos
3) Bias (cesgo o umbral)
4) Sumatoria
5) Función de Activación
6) Salida



Cómo funciona

- Realizamos una suma ponderada de las entradas, los pesos y el bias

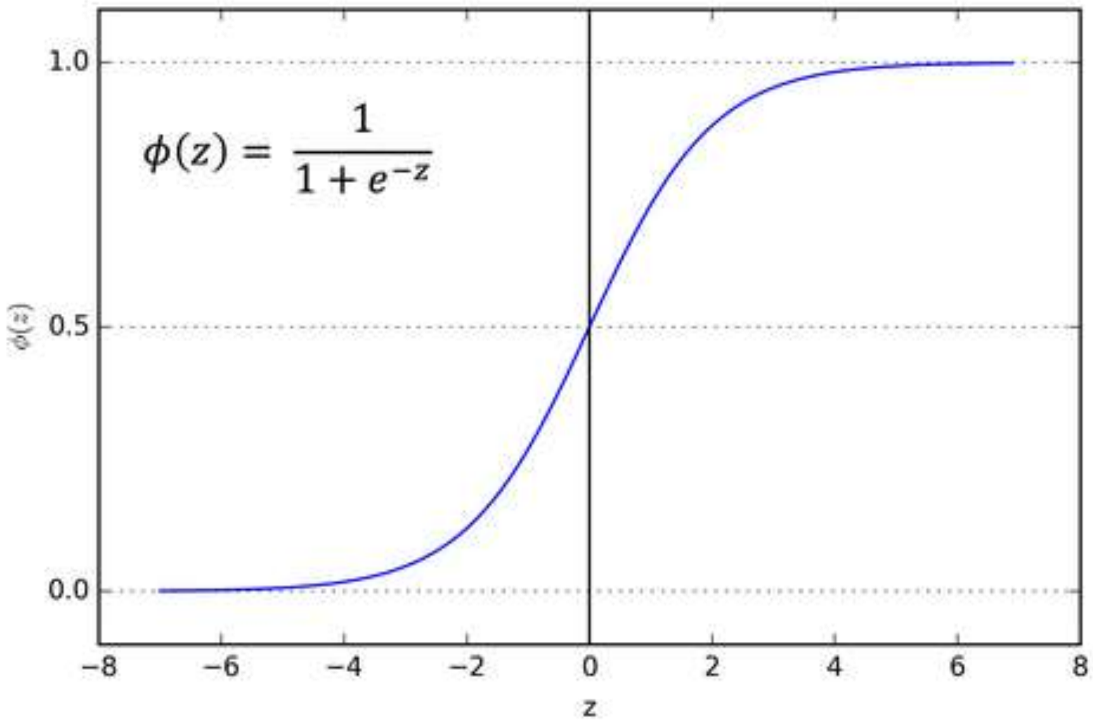
$$SumaPonderada = \sum_{i=1}^{n} x_i w_i + b$$

- Aplicamos una función de activación

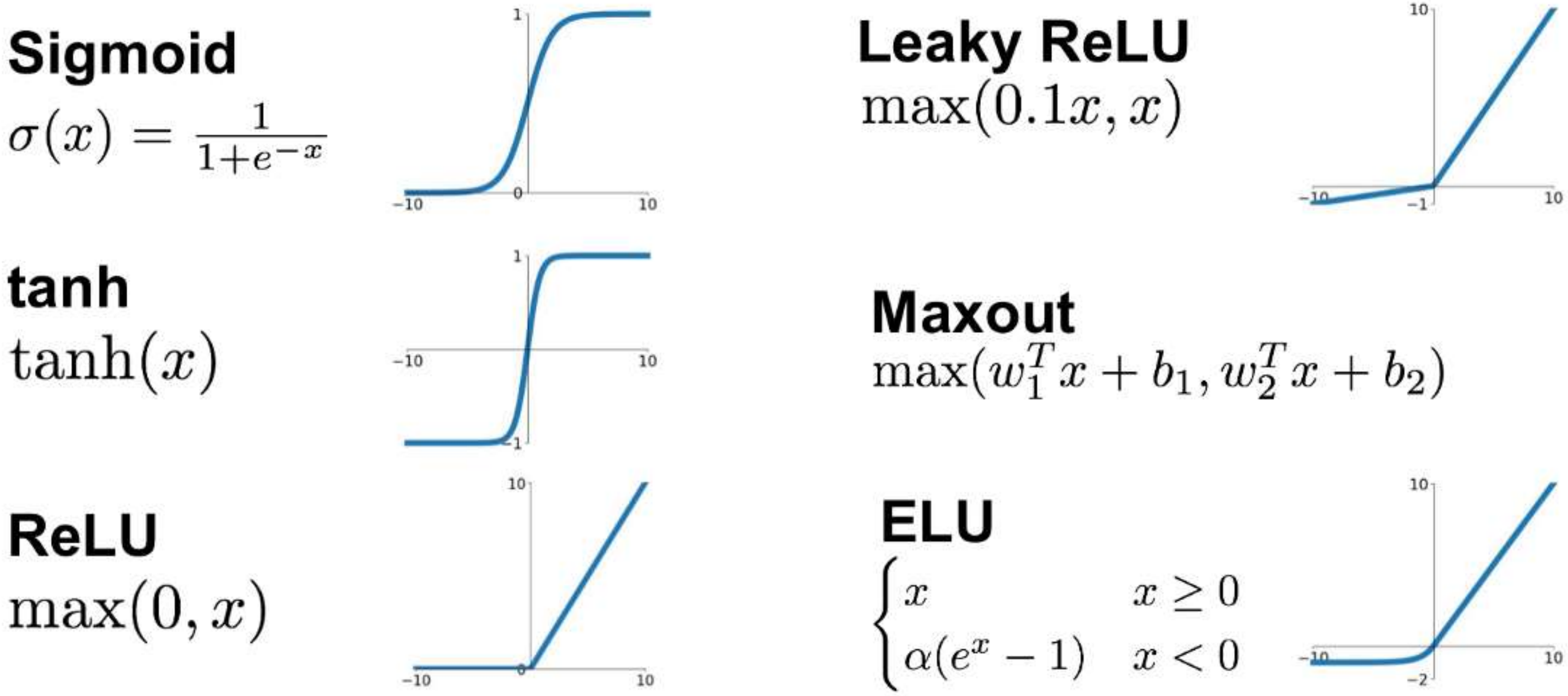
$$\phi(SumaPonderada)$$

Función de Activación

- También conocido como función de transferencia
- Usado para determinar la salida de la neurona
- No necesariamente debe ir de 0 a 1
- Hay muchas

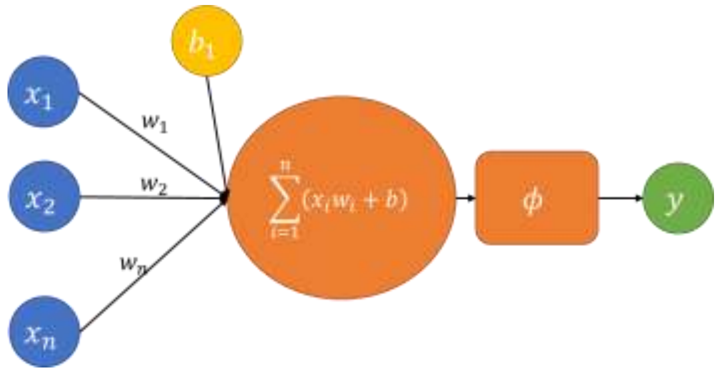


Funciones de Activación



Toman un valor de entrada y aplican una transformación no lineal

Perceptrón

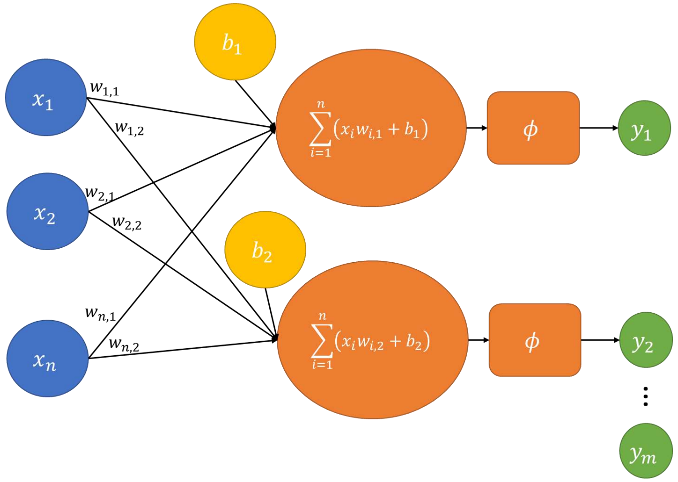


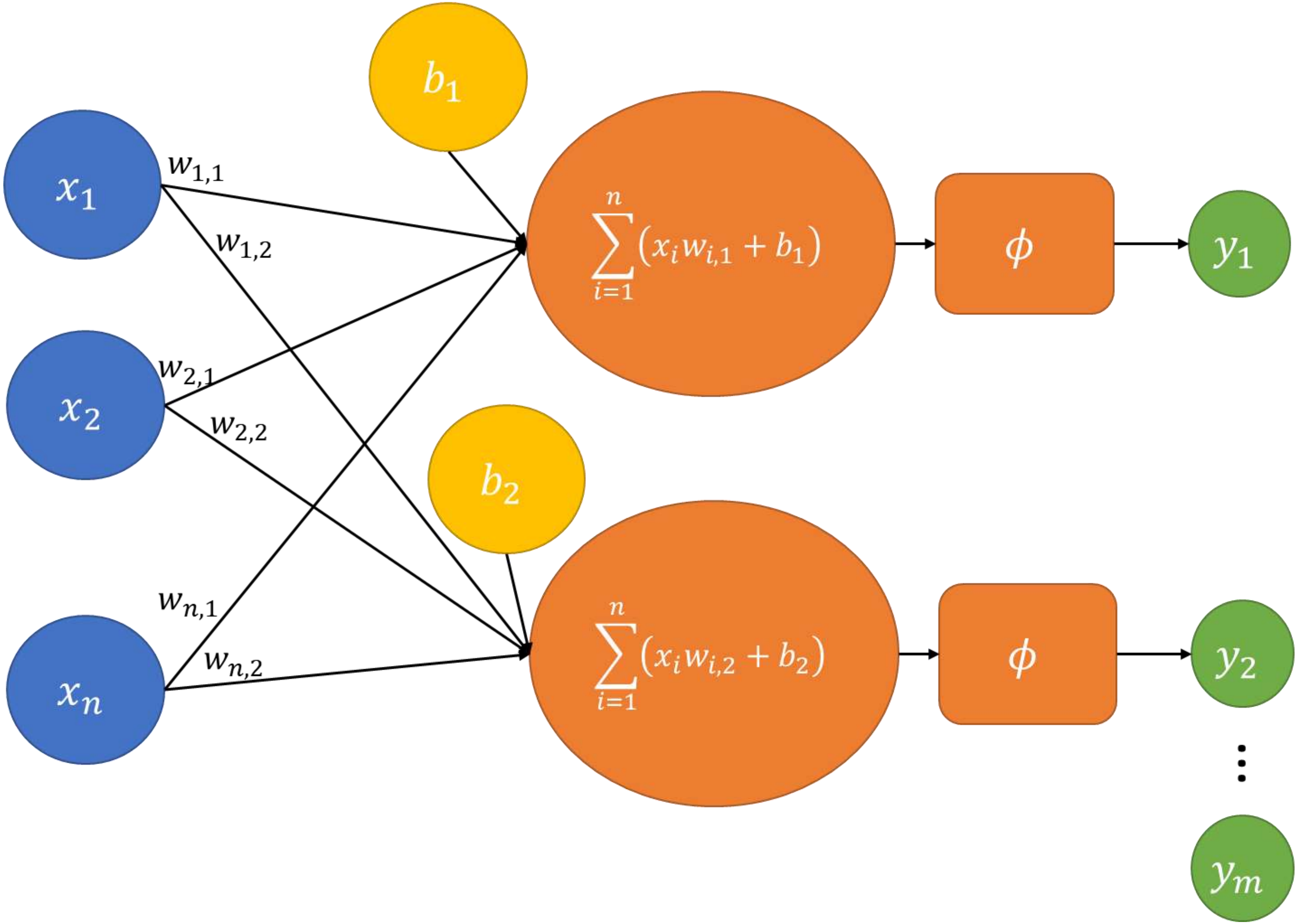
Matemáticamente:

$$y = \phi\left(\sum_{i=1}^{n} x_i w_i + b\right)$$

Si tenemos varias salidas…

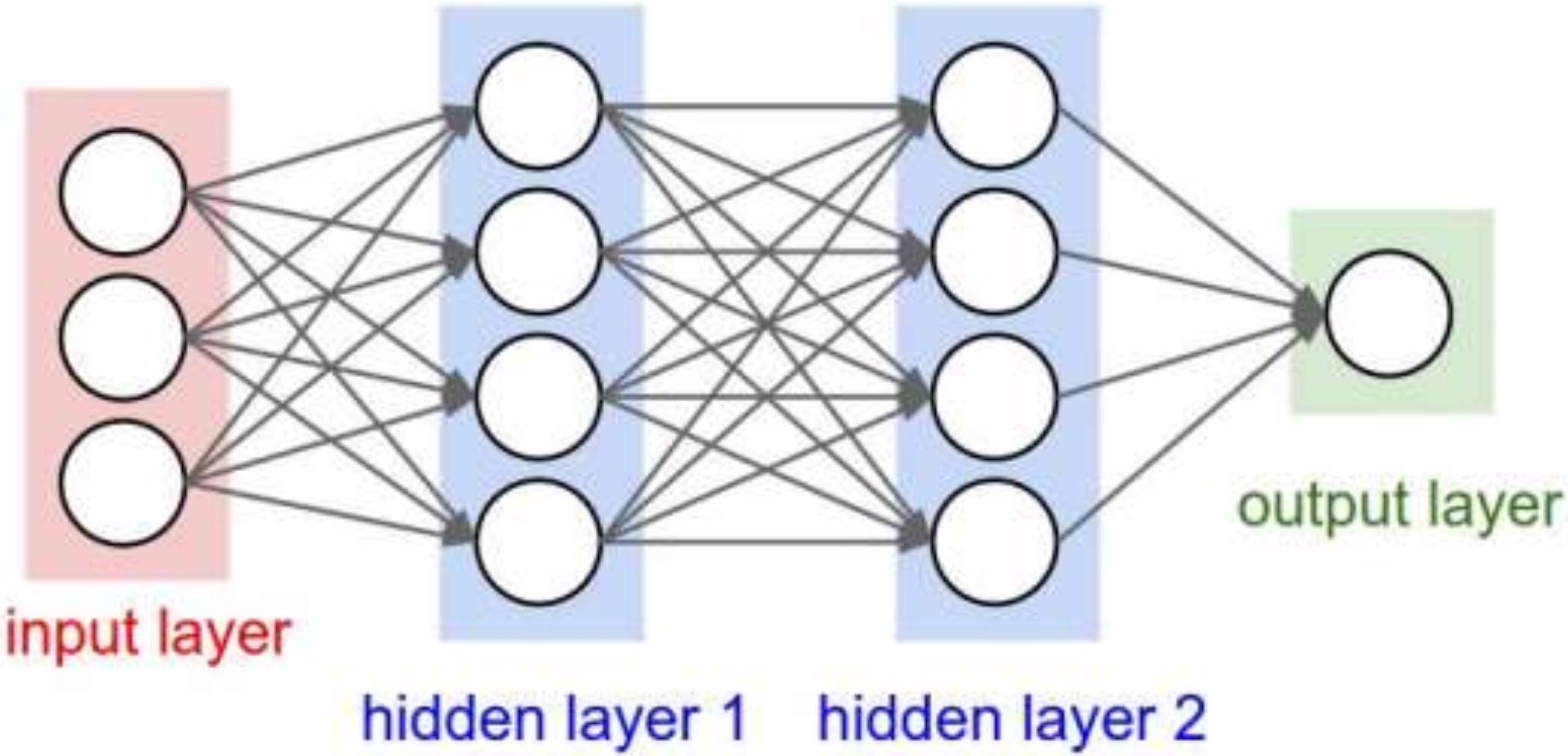
$$y_k = \phi\left(\sum_{i=1}^{n} x_i w_{i,k} + b_k\right)$$



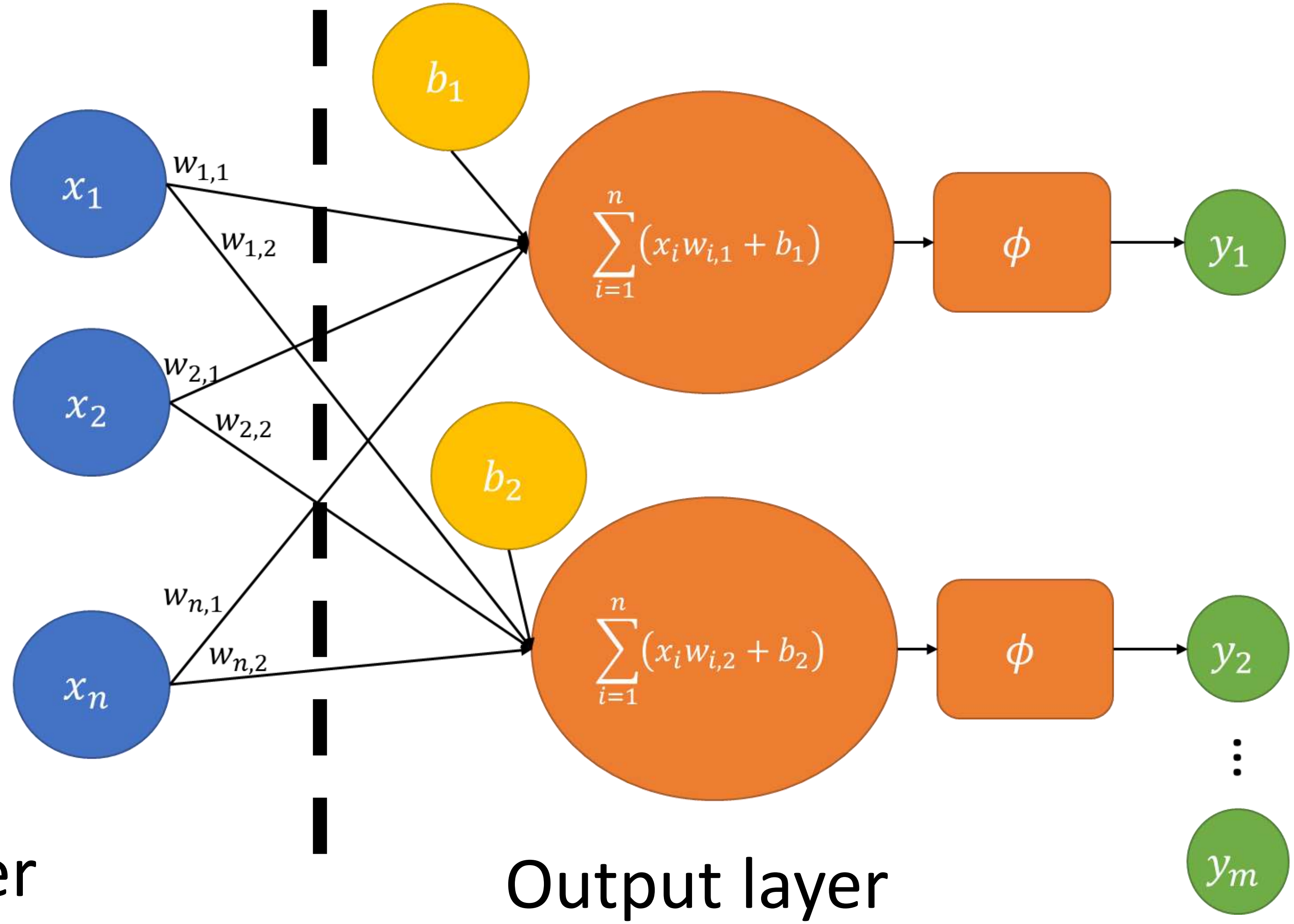

b_1
x_1
$w_{1,1}$
$w_{1,2}$
$\sum_{i=1}^{n}(x_i w_{i,1} + b_1)$
ϕ
y_1
x_2
$w_{2,1}$
$w_{2,2}$
b_2
$w_{n,1}$
x_n
$w_{n,2}$
$\sum_{i=1}^{n}(x_i w_{i,2} + b_2)$
ϕ
y_2
$\vdots$
y_m

Perceptrón Multicapa
Feed Forward Neural Network

Agregando capas


input layer
hidden layer 1
hidden layer 2
output layer

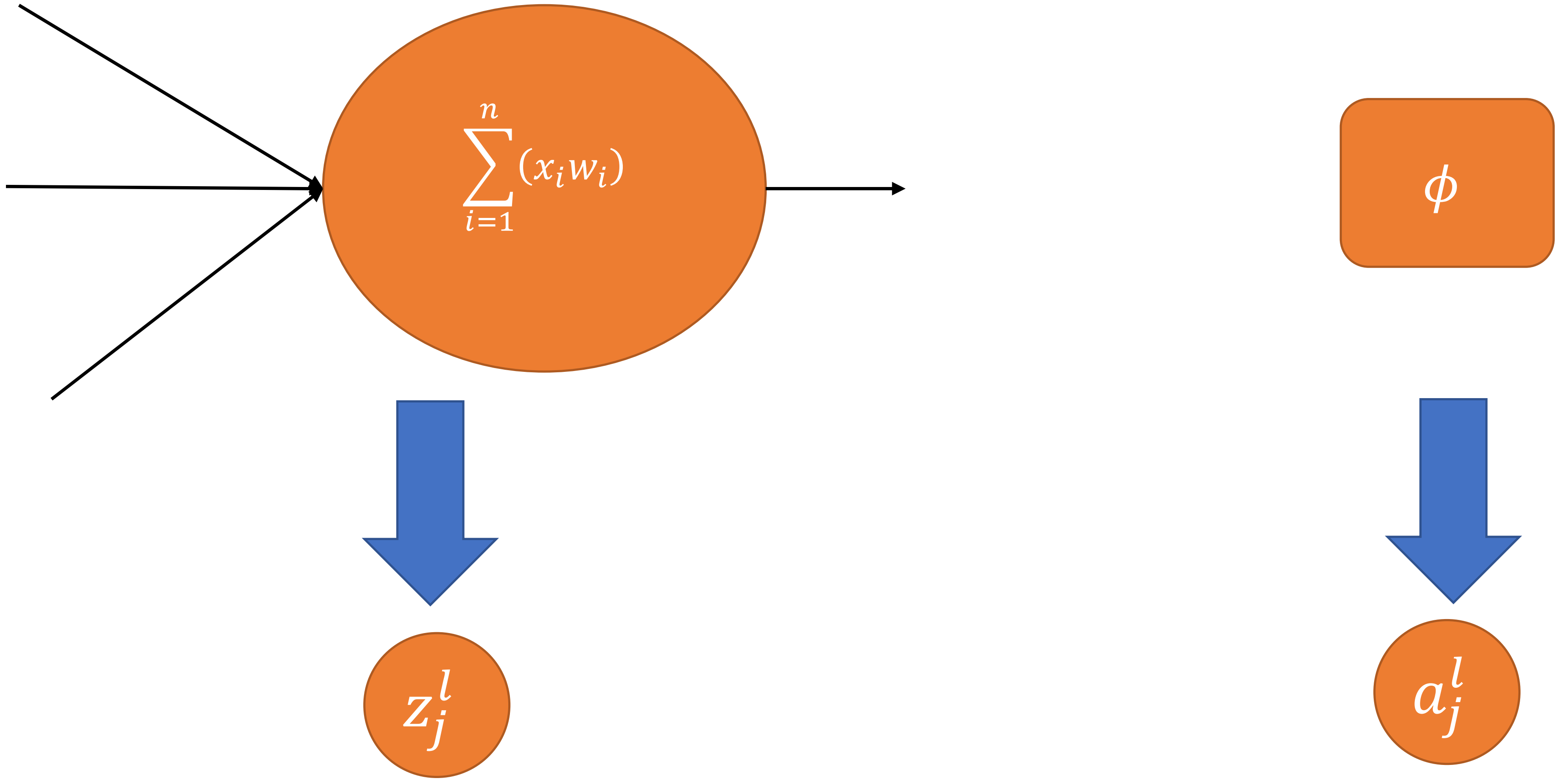
Cambiando notación

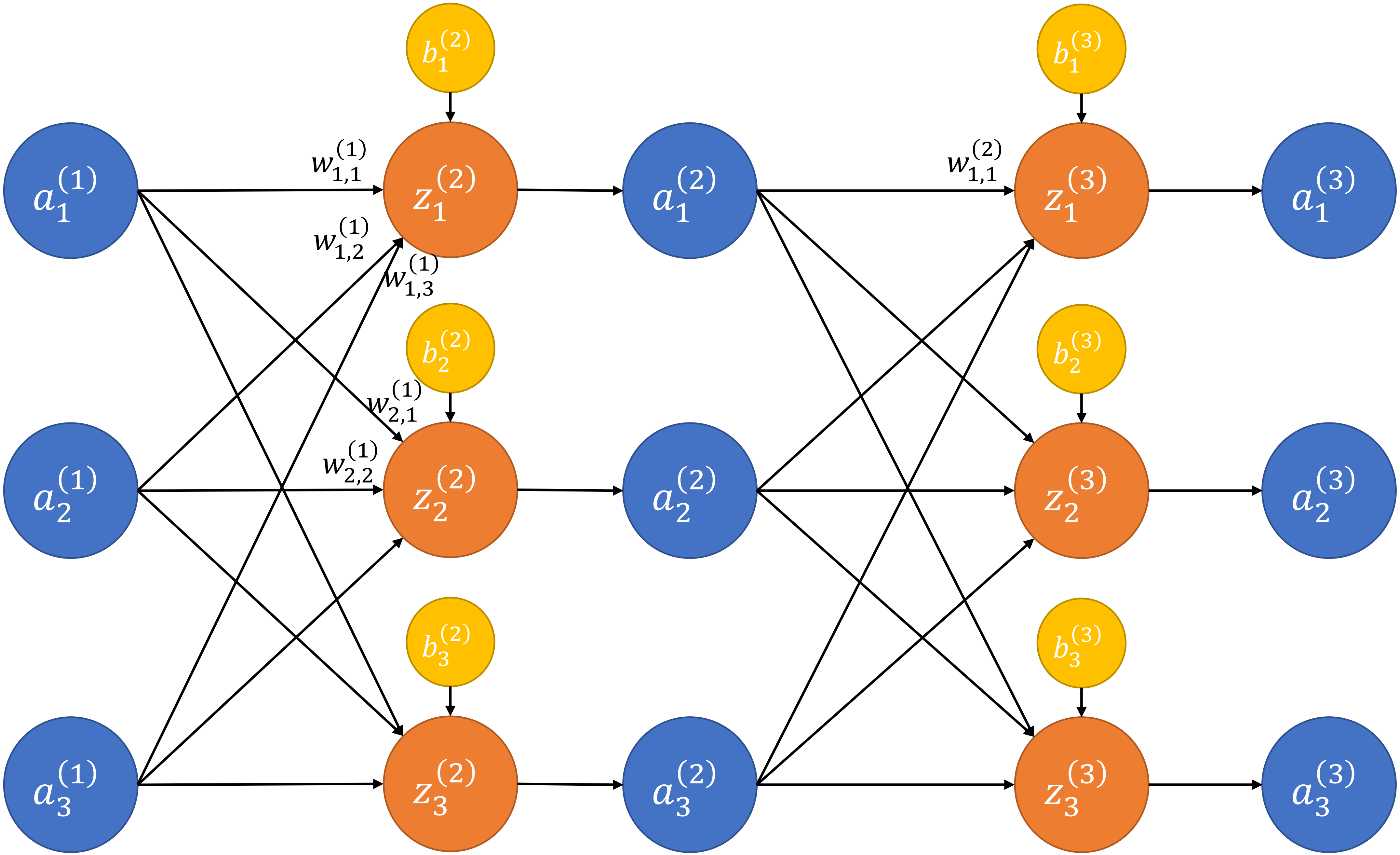


Input layer

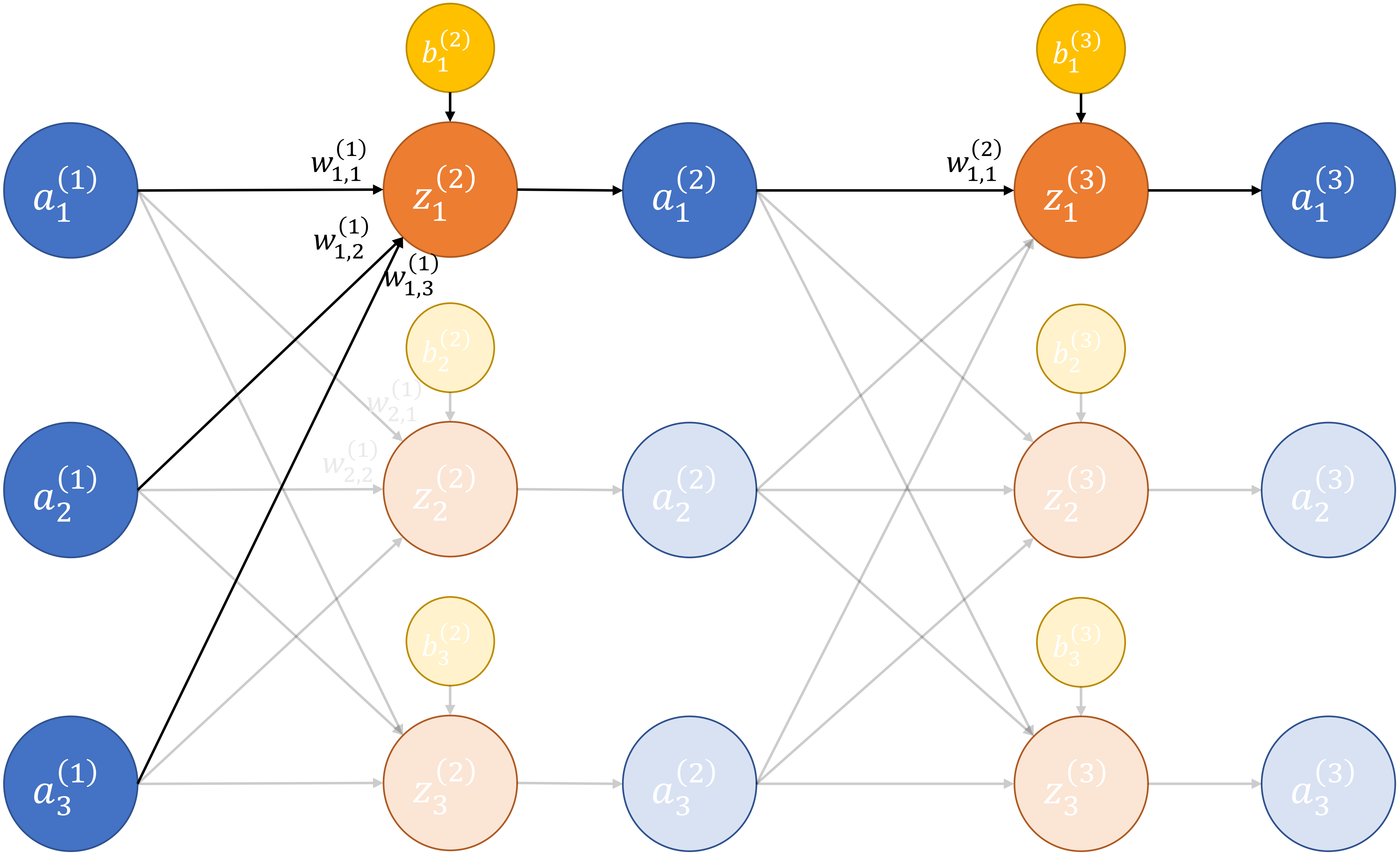
Output layer

Cambiando notación

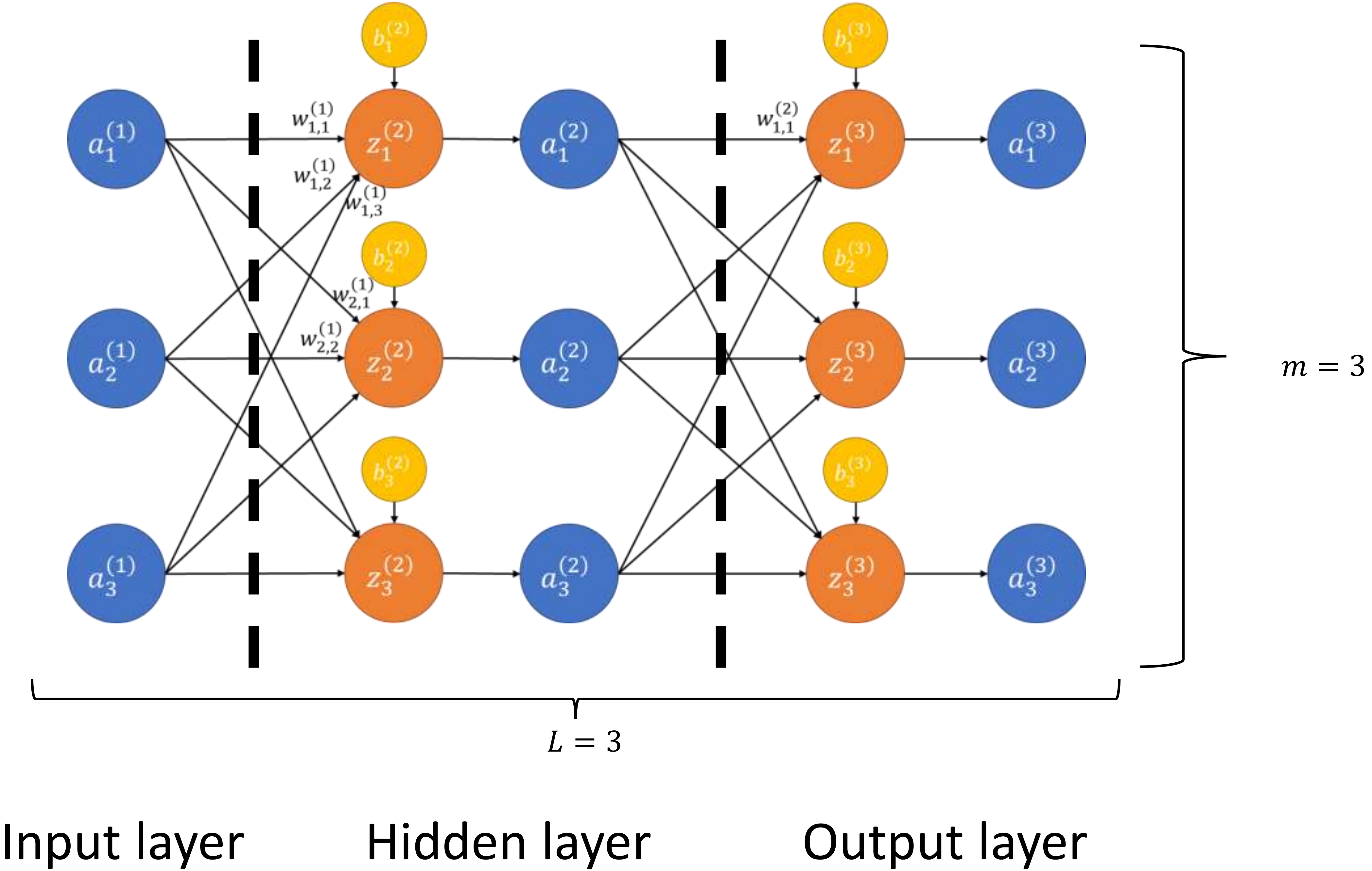




$a_1^{(1)}$
$a_2^{(1)}$
$a_3^{(1)}$
$w_{1,1}^{(1)}$
$w_{1,2}^{(1)}$
$w_{1,3}^{(1)}$
$w_{2,1}^{(1)}$
$w_{2,2}^{(1)}$
$b_1^{(2)}$
$b_2^{(2)}$
$b_3^{(2)}$
$z_1^{(2)}$
$z_2^{(2)}$
$z_3^{(2)}$
$a_1^{(2)}$
$a_2^{(2)}$
$a_3^{(2)}$
$w_{1,1}^{(2)}$
$b_1^{(3)}$
$b_2^{(3)}$
$b_3^{(3)}$
$z_1^{(3)}$
$z_2^{(3)}$
$z_3^{(3)}$
$a_1^{(3)}$
$a_2^{(3)}$
$a_3^{(3)}$



$b_1^{(2)}$
$b_1^{(3)}$
$a_1^{(1)}$
$w_{1,1}^{(1)}$
$z_1^{(2)}$
$a_1^{(2)}$
$w_{1,1}^{(2)}$
$z_1^{(3)}$
$a_1^{(3)}$
$w_{1,2}^{(1)}$
$w_{1,3}^{(1)}$
$b_2^{(2)}$
$b_2^{(3)}$
$w_{2,1}^{(1)}$
$w_{2,2}^{(1)}$
$a_2^{(1)}$
$z_2^{(2)}$
$a_2^{(2)}$
$z_2^{(3)}$
$a_2^{(3)}$
$b_3^{(2)}$
$b_3^{(3)}$
$a_3^{(1)}$
$z_3^{(2)}$
$a_3^{(2)}$
$z_3^{(3)}$
$a_3^{(3)}$



$a_1^{(1)}$
$a_2^{(1)}$
$a_3^{(1)}$
$w_{1,1}^{(1)}$
$w_{1,2}^{(1)}$
$w_{1,3}^{(1)}$
$w_{2,1}^{(1)}$
$w_{2,2}^{(1)}$
$b_1^{(2)}$
$b_2^{(2)}$
$b_3^{(2)}$
$z_1^{(2)}$
$z_2^{(2)}$
$z_3^{(2)}$
$a_1^{(2)}$
$a_2^{(2)}$
$a_3^{(2)}$
$w_{1,1}^{(2)}$
$b_1^{(3)}$
$b_2^{(3)}$
$b_3^{(3)}$
$z_1^{(3)}$
$z_2^{(3)}$
$z_3^{(3)}$
$a_1^{(3)}$
$a_2^{(3)}$
$a_3^{(3)}$
$m = 3$
$L = 3$
Input layer
Hidden layer
Output layer

Notación Matemática

- L número de capas en la red donde la capa 1 es la capa de entrada y la capa L es la capa de salida
- m el "ancho" de la red (puede variar de capa en capa)
- $w_{j,k}^{l}$ el peso de la conexión entre la k-esima unidad de la capa $l-1$ hacia la j-esima unidad de la capa l
- b_j^l "bias" de la j-esima unidad de la capa l
- $z_j^l = \sum_k w_{j,k}^l a_k^{l-1} + b_j^l$ suma ponderada de los inputs de j en la capa l
- $a_j^l = \phi(z_j^l)$ "Activación" de la unidad j en la capa l donde ϕ es la "función de activación"

Entrenando la red neuronal

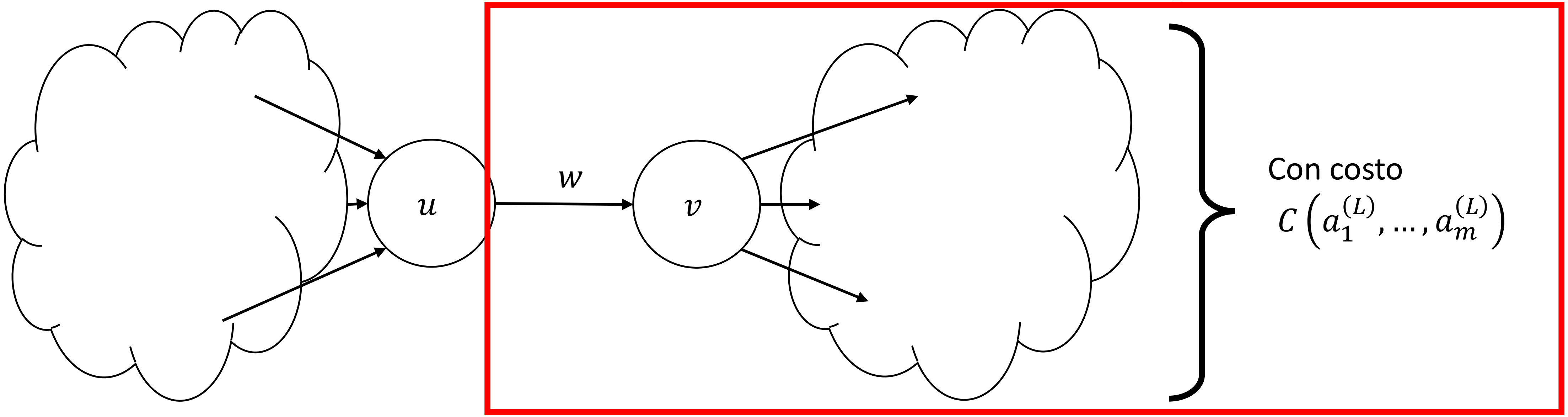
Nuestro Objetivo

- Los parámetros que queremos encontrar en la red son
  - $w_{j,k}^{l}$ los pesos
  - b_{j}^{l} los bias
- Problema de optimización
- Gradient Descent
- Debemos primero encontrar la gradiente para mover los valores

Idea general

Entradas
$a_1^{(1)}, \dots, a_m^{(1)}$

Salidas
$a_1^{(L)}, \dots, a_m^{(L)}$



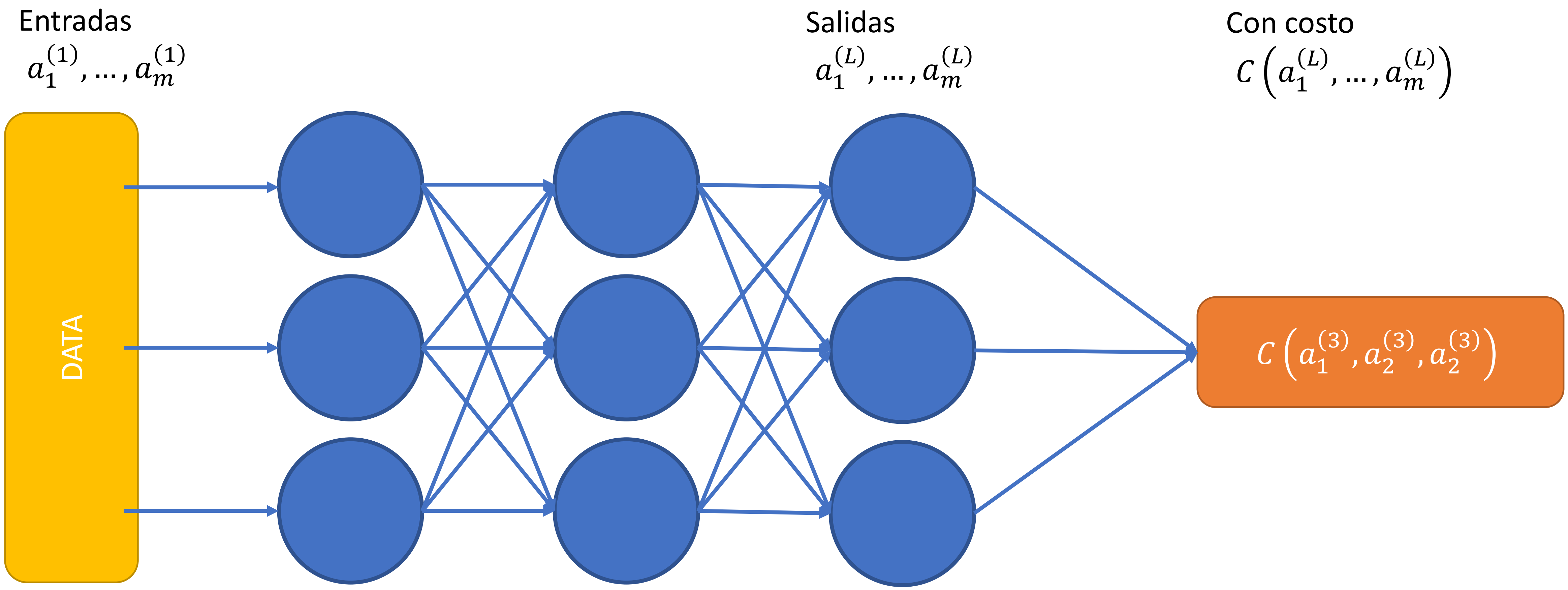
Para optimizar los pesos y bias en una red, usamos el gradient descent.

$$w_{nuevo} = w_{antiguo} - LearningRate * \frac{\partial C}{\partial w}$$

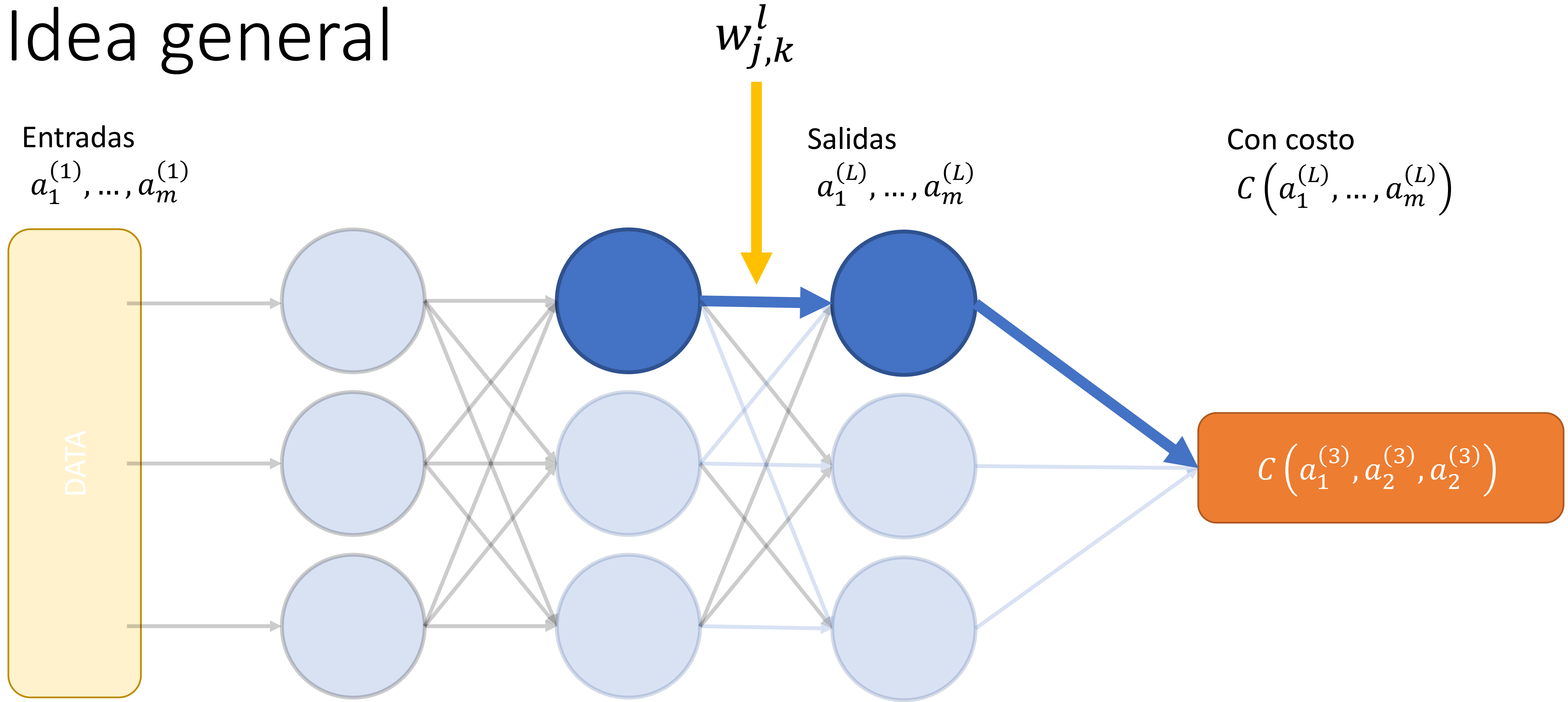
Para ello, debemos usar la regla de la cadena y encontrar la gradiente

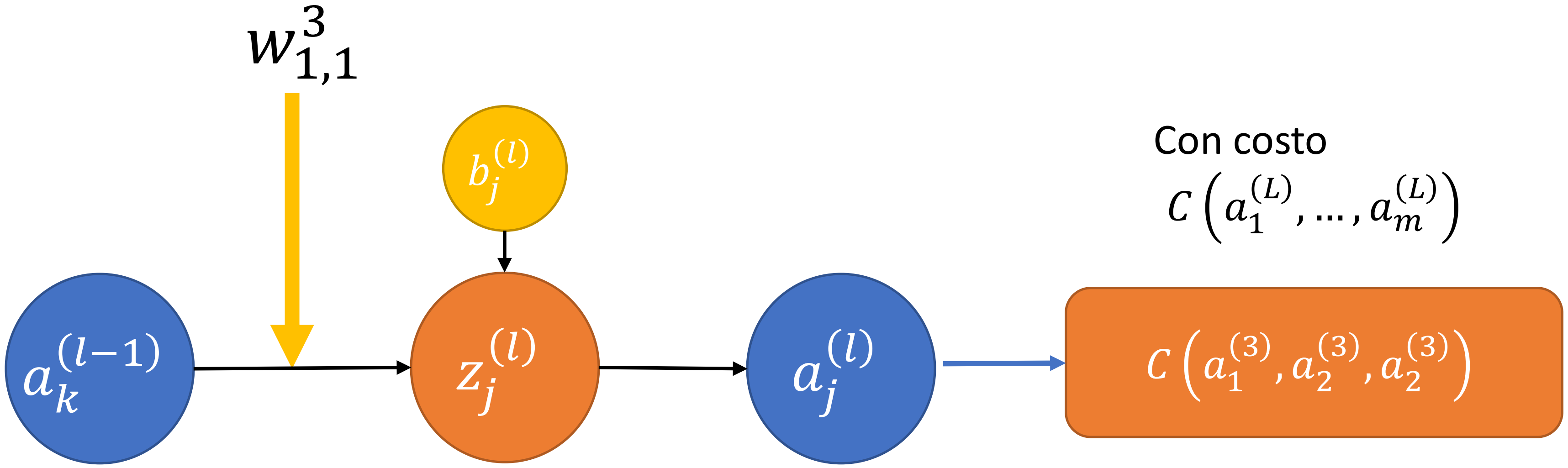
$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial v} \cdot \frac{\partial v}{\partial w}$$

Idea general

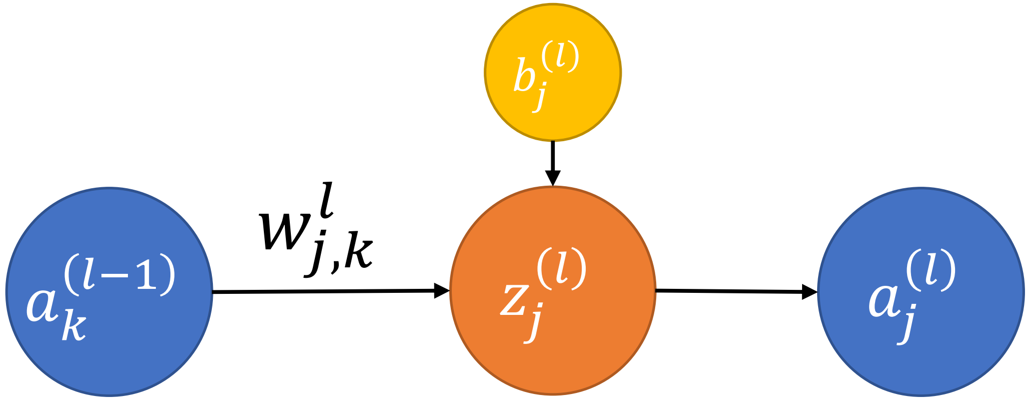


Idea general





$$z_j^l = \sum_k w_{j,k}^l a_k^{l-1} + b_j^l$$
$$a_j^l = \phi\left(z_j^l\right)$$



$$z_j^l = \sum_k w_{j,k}^l a_k^{l-1} + b_j^l$$
$$a_j^l = \phi\left(z_j^l\right)$$

Derivada de peso

$$\frac{\partial C}{\partial w_{j,k}^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{j,k}^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot a_k^{l-1}$$

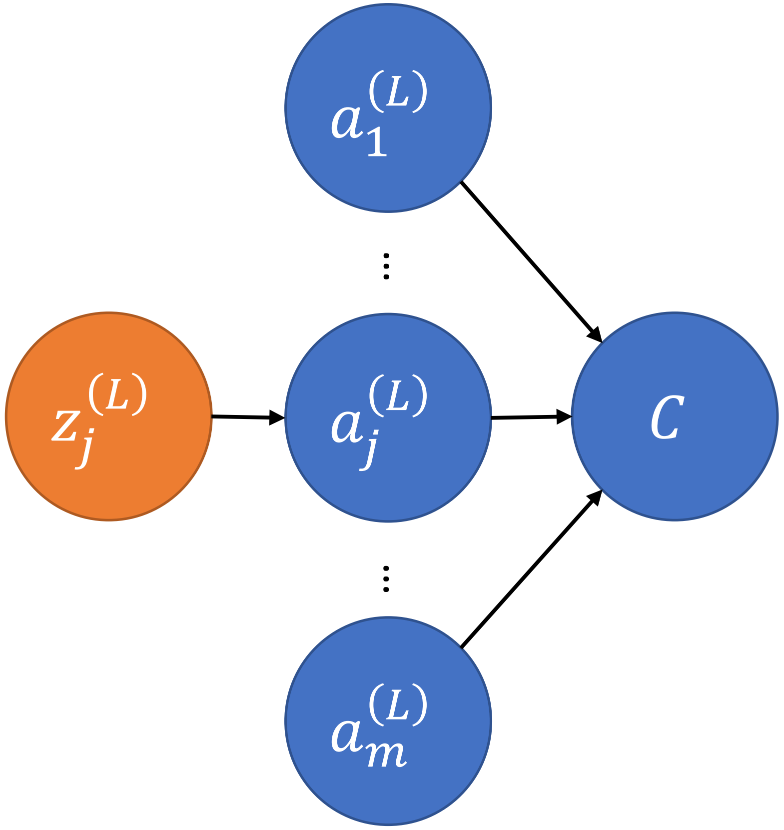
Derivada de bias

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot 1$$

Podemos obtener $\frac{\partial C}{\partial w_{j,k}^{(l)}}$ y $\frac{\partial C}{\partial b_j^{(l)}}$ si es que conocemos $\delta_j^l \coloneqq \frac{\partial C}{\partial z_j^{(l)}}$

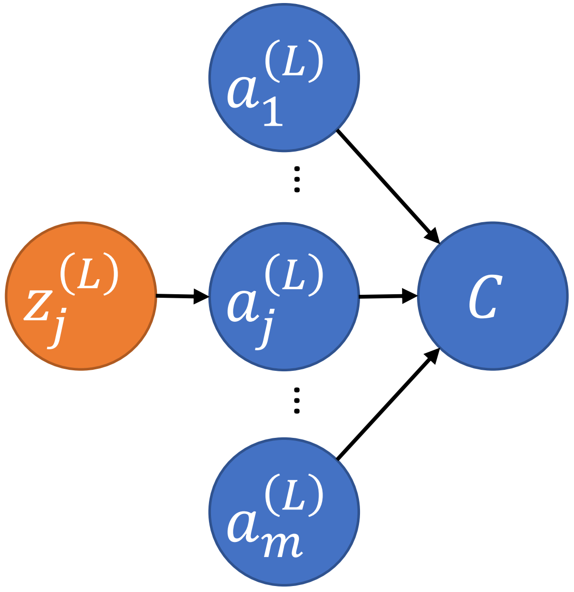
Conocido también como “<u>gradiente local</u>”

Gradiente local en la capa de salida



- $a_j^L = \phi(z_j^L)$
- La gradiente local para la capa de salida:
- $\delta_j^L = \dfrac{\partial C}{\partial z_j^{(L)}}$ por definición
- $\dfrac{\partial C}{\partial z_j^{(l)}} = \dfrac{\partial C}{\partial a_j^{(L)}} \cdot \dfrac{\partial a_j^{(L)}}{\partial z_j^{(L)}}$ regla cadena
- $= \dfrac{\partial C}{\partial a_j^{(L)}} \cdot \phi'\left(z_j^{(L)}\right)$

Gradiente local en la capa de salida



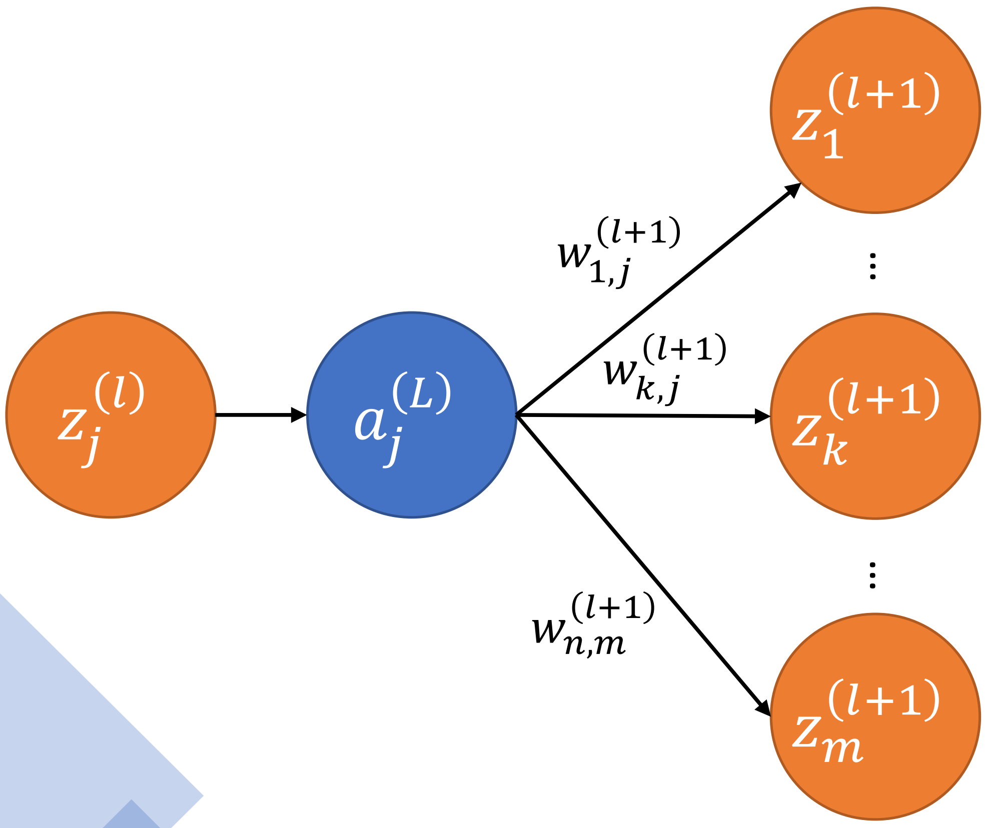
- $\dfrac{\partial C}{\partial a_j^{(L)}} \cdot \phi'\left(z_j^{(L)}\right)$

- $\dfrac{\partial C}{\partial a_j^{(L)}}$ depende de la función de costo.

- Si usamos la función mínimos cuadrados:

- $C\left(a_1^{(L)}, \dots, a_m^{(L)}\right) = \sum_{k=1}^{m} \left(y_k - a_k^{(L)}\right)^2$
- $\dfrac{\partial C}{\partial a_j^{(L)}} = a_j^{(L)} - y_j$

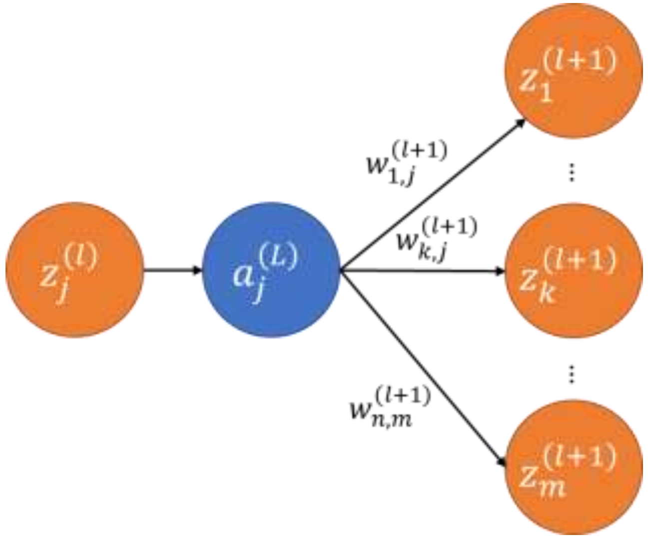
Gradiente local en capa oculta



$$z_k^{l+1} = \sum_r w_{r,k}^{l+1} a_r^l + b_k^{l+1}$$
$$a_j^l = \phi\left(z_j^l\right)$$

Nuevamente, debemos encontrar δ_j^l

Gradiente local en capa oculta



$$z_k^{l+1} = \sum_r w_{r,k}^{l+1} a_r^l + b_k^{l+1}$$
$$a_j^l = \phi(z_j^l)$$

$\delta_j^l = \frac{\partial C}{\partial z_j^{(l)}}$ por definición de la gradiente local

$= \frac{\partial C}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}}$ por la regla cadena

$= \left(\sum_k \frac{\partial C}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}} \right) \cdot \phi'(z_j^l)$ por la regla cadena

$= \phi'(z_j^l) \cdot \sum_k \delta_k^{l+1} \cdot w_{k,j}^{l+1}$ por la definición de gradiente local δ_k^{l+1}

Resumiendo

Para todos los pesos y bias

$$\frac{\partial C}{\partial w_{j,k}^{(l)}} = \delta_j^l \cdot a_k^{l-1} \qquad\qquad \frac{\partial C}{\partial b_j^{(l)}} = \delta_j^l$$

Donde la gradiente local δ_j^l es

$$\delta_j^l = \begin{cases} \phi'\left(z_j^{(L)}\right) \cdot \dfrac{\partial C}{\partial a_j^{(L)}} \, \boldsymbol{si\ es\ la\ capa\ de\ salida} \\ \phi'\left(z_j^l\right) \cdot \displaystyle\sum_k \delta_k^{l+1} \cdot w_{k,j}^{l+1} \, \boldsymbol{si\ es\ una\ capa\ oculta} \end{cases}$$

Pasando Todo a Matrices

Suma ponderada

$$z^{(l)} = \left(z_1^{(l)}, \dots, z_m^{(l)}\right)$$

$$= \left(\sum_{k=1}^{m} w_{1k}^{(l)} a_k^{(l-1)} + b_1^{(l)}, \dots, \sum_{k=1}^{m} w_{mk}^{(l)} a_k^{(l-1)} + b_m^{(l)}\right)$$

$$= w^{(l)} a^{(l-1)} + b^{(l)}$$

Pasando Todo a Matrices

Función de Activación

$$a^{(l)} = \left(a_1^{(l)}, \dots, a_m^{(l)}\right)$$

$$= \left(\phi\left(z_1^{(l)}\right), \dots, \phi\left(z_m^{(l)}\right)\right)$$

$$= \phi\left(z^{(l)}\right)$$

Pasando Todo a Matrices

Gradiente Local a la salida

$$\delta^L = \left(\delta_1^{(L)}, \dots, \delta_m^{(L)}\right)$$

$$= \left(\frac{\partial C}{\partial a_1^{(L)}} \cdot \phi'\left(z_1^{(L)}\right), \dots, \frac{\partial C}{\partial a_m^{(L)}} \cdot \phi'\left(z_m^{(L)}\right)\right)$$

$$= \nabla_{a^{(L)}} C \cdot \phi'\left(z^{(L)}\right)$$

Pasando Todo a Matrices

Gradiente Local en una capa oculta

$$\delta^l = \left(\delta_1^{(l)}, \dots, \delta_m^{(l)}\right)$$

$$= \left(\phi'\left(z_1^{(l)}\right) \cdot \sum_k \delta_k^{l+1} \cdot w_{k,1}^{l+1}, \dots, \phi'\left(z_m^{(l)}\right) \cdot \sum_k \delta_k^{l+1} \cdot w_{k,m}^{l+1}\right)$$

$$= \phi'\left(z^{(l)}\right) \odot \left(w^{(l+1)}\right)^T \delta^{l+1}$$

Backpropagation

Algoritmo de Backpropagation resumido

Tomamos un item (x, y) del dataset

1) Definimos la activación de la capa de entrada como $a^{(1)} = x$
2) Para cada capa realizar feed forward
3) Calcular la gradiente local para la salida
4) Realizar backpropagation de las gradientes locales para las capas ocultas
5) Retornar las derivadas parciales de pesos y bias

1) Inicializar

- Definimos la activación de la capa de entrada
  - $a^{(1)} = x_i$
  - x_i es el i-ésimo item de nuestro dataset de tamaño n
- Inicializamos los pesos $w^{(l)}$ y bias $b^{(l)}$
  - Puede ser 0
  - Puede ser aleatorio

2) Feed Forward

- Realizamos el cálculo de nuestra predicción
- Para cada capa:

$$z^{(l)} = w^{(l)}a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = \phi\left(z^{(l)}\right)$$

- Hasta llegar a la capa de la salida $a^{(L)}$
- Determinamos nuestro costo por item y el costo promedio

$$C^{(i)} = {}^{1}\!/_{2}\left(y_i - a^{(L)}\right)^2$$

$$C = {}^{1}\!/_{b}\sum_{i=1}^{b} C^{(i)}$$

3 y 4) Backpropagation

- La gradiente "promedio" por cada item

$$\frac{\partial C}{\partial w^{(l)}} = \frac{1}{b}\sum_{i=1}^{b}\frac{\partial C^{(i)}}{\partial w^{(l)}}$$

$$\frac{\partial C}{\partial b^{(l)}} = \frac{1}{b}\sum_{i=1}^{b}\frac{\partial C^{(i)}}{\partial b^{(l)}}$$

5) Determinar los nuevos pesos y bias

- Determinar los nuevos pesos y bias (update rule)

$$w^{(l)} \leftarrow w^{(l)} - \eta \frac{\partial C}{\partial w^{(l)}}$$

$$b^{(l)} \leftarrow b^{(l)} - \eta \frac{\partial C}{\partial b^{(l)}}$$

Variantes

- Batch GD $b = n$
- Mini-batch GD $b < n$ (usualmente 20 a 100)
- Stochastic GD $b = 1$

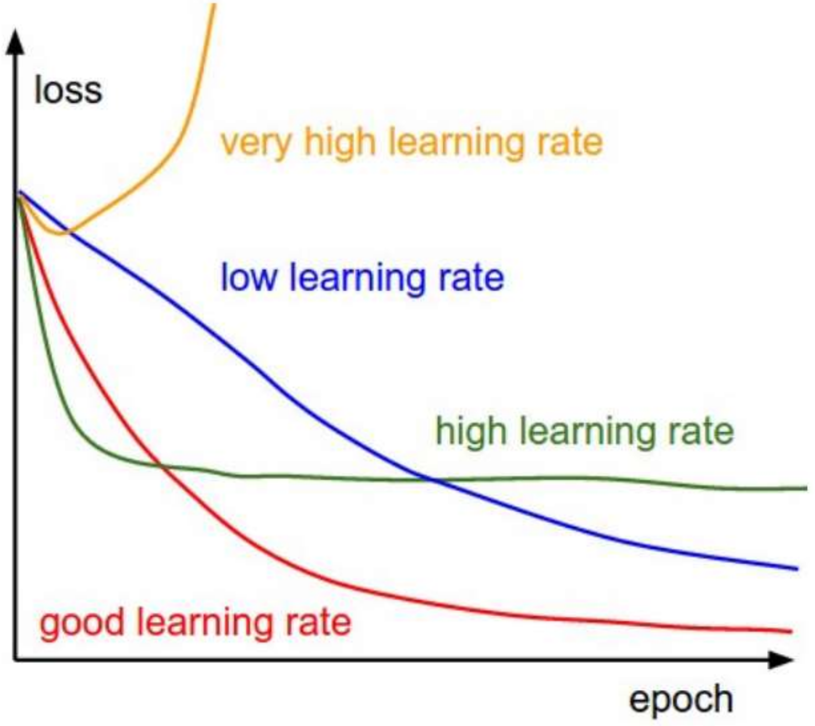
- η el learning rate puede ser fijo
- Puede ser adaptativo
- Con momentum

Problemas

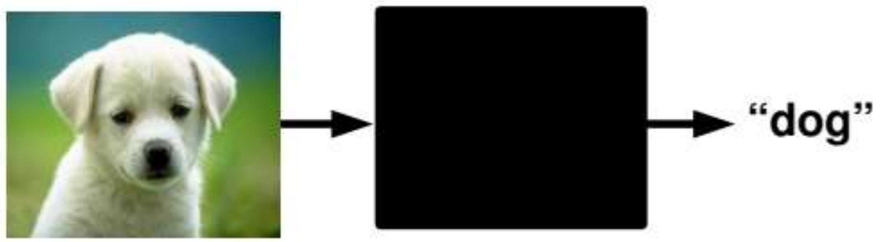
- Si el modelo es muy complejo es propenso al overfitting

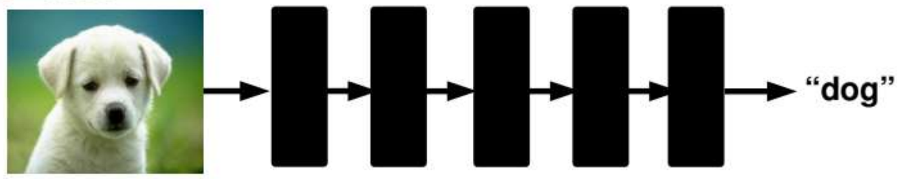
Solución

- Reducir el número de capas ocultas
- Regularización
- Early stopping



Deep?





Conclusiones

Redes Neuronales

Basados en modelo perceptrón

Algoritmo Feedforward

Gradient Descent usando Backpropagation

Video Recomendado

- https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi