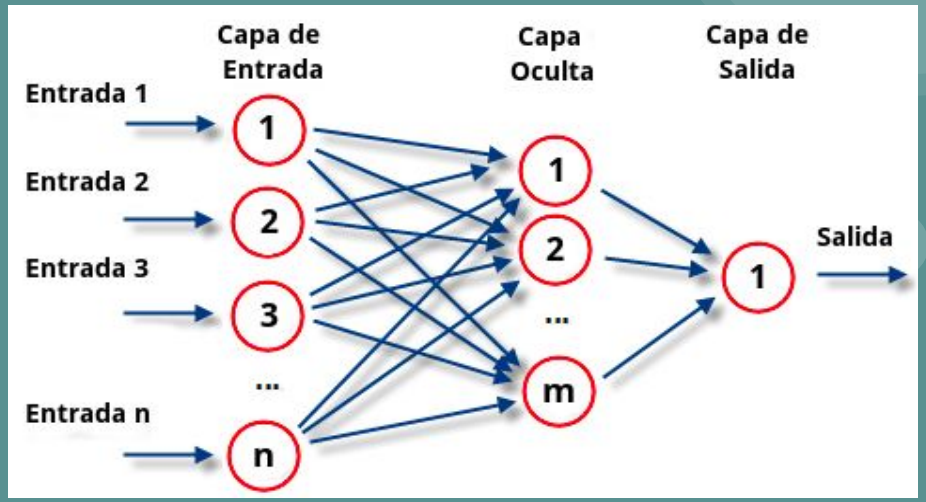
Implementación de una red neuronal artificial en c++.

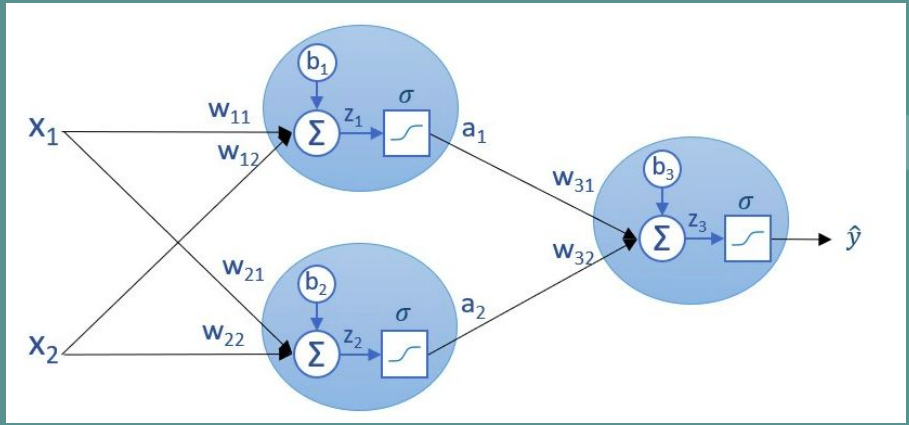
Ana María Correa Castrillón
Edwin Dair Zapata Duque

¿ Qué saber de una red neuronal?

- ➔ Es un modelo computacional inspirado en las conexiones en red que forman las neuronas cerebrales.
- ➔ Las neuronas están dispuestas en capas: una capa de entrada, una o varias capas ocultas y una capa de salida.
- ➔ El peso de cada conexión entre neuronas determina la importancia en el cálculo de la salida.

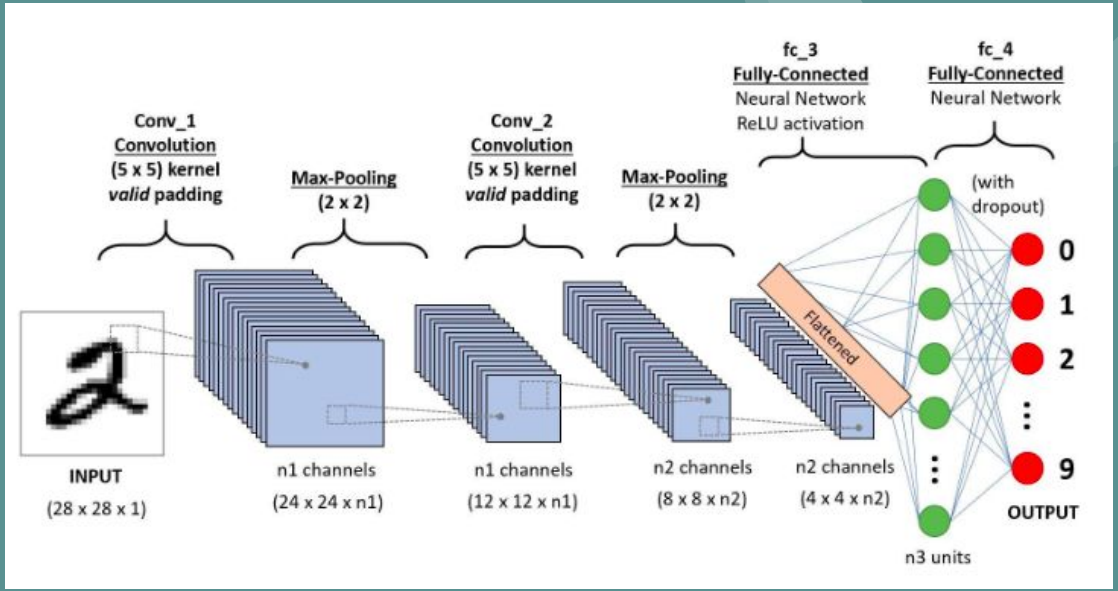


- ➔ Durante el entrenamiento de la red neuronal, los pesos se ajustan iterativamente para minimizar la diferencia entre las salidas de la red y los valores deseados.
- ➔ La capacidad de la red neuronal para aprender y realizar tareas proviene de su aptitud para reconocer patrones y relaciones en los datos.
- ➔ Las redes neuronales se utilizan en una amplia gama de aplicaciones y en muchas áreas que requieren el análisis y procesamiento de grandes cantidades de datos.



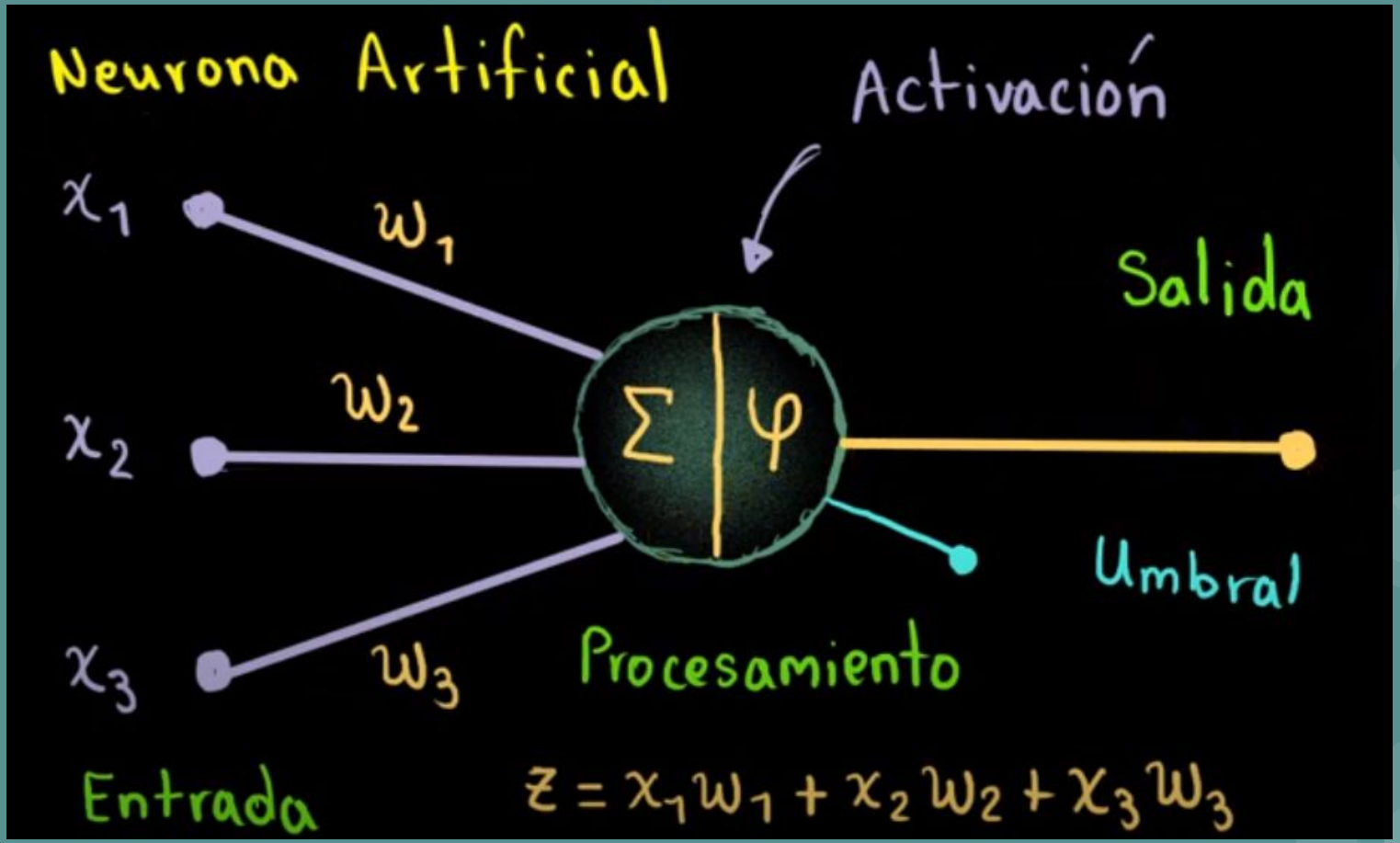
Red neuronal convolucional

Es un tipo de red neuronal artificial donde las neuronas corresponden a campos receptivos de una manera similar que lo hacen las neuronas de la corteza visual del cerebro.



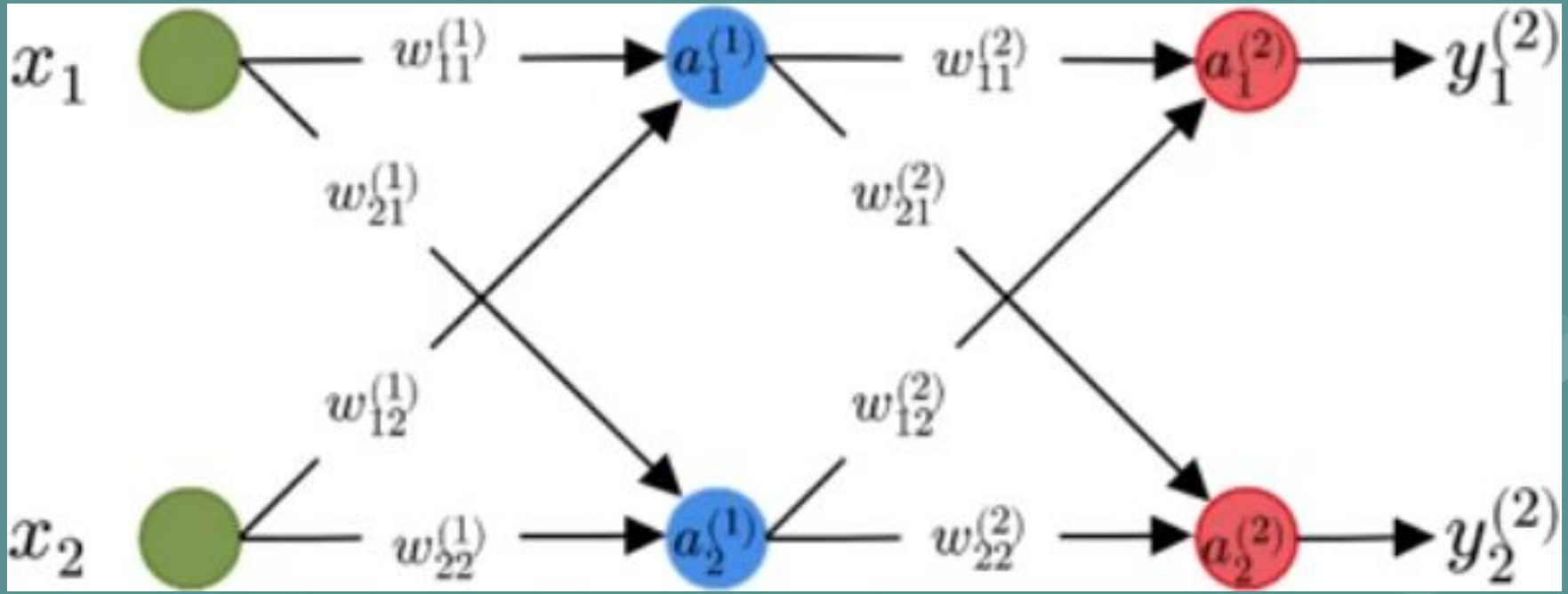
Perceptrón

- Consta de una única capa, por lo que es el modelo más simple.
- Permitió sentar las bases para las neuronas multicapas.
- Implementación:
  - Se reciben los datos a la capa de entrada.
  - Cada uno de los datos se multiplica por su peso.
  - Se suman los resultados del punto anterior.
  - Dicha suma se pasa a través de la función de activación. En este caso es una función sigmoide que produce valores continuos entre 0 y 1.
  - La salida de la función de activación, es la salida final del perceptrón.


Neurona Artificial
Activación
x_1
w_1
Salida
w_2
x_2
Σ
φ
Umbral
x_3
w_3
Procesamiento
Entrada
$z = x_1 w_1 + x_2 w_2 + x_3 w_3$

¿ Cómo se entrena una red neuronal?

- Datos de entrenamiento: Datos de entrada, datos de salida.
- Ajuste de los pesos.
- Pasos para entrenar la red:
  - Se preparan los datos.
  - Se inicializa la red generando pesos aleatorios.
  - Se realiza el forward propagation.
  - Se calcula la función error, comparando la salida de la red con cada dato de salida dado en el entrenamiento.
  - Se realiza el backpropagation, en donde se hace uso del gradiente descendente.
  - Se actualizan los pesos.
  - Se repite el proceso hasta que se llegue al número de épocas dado.

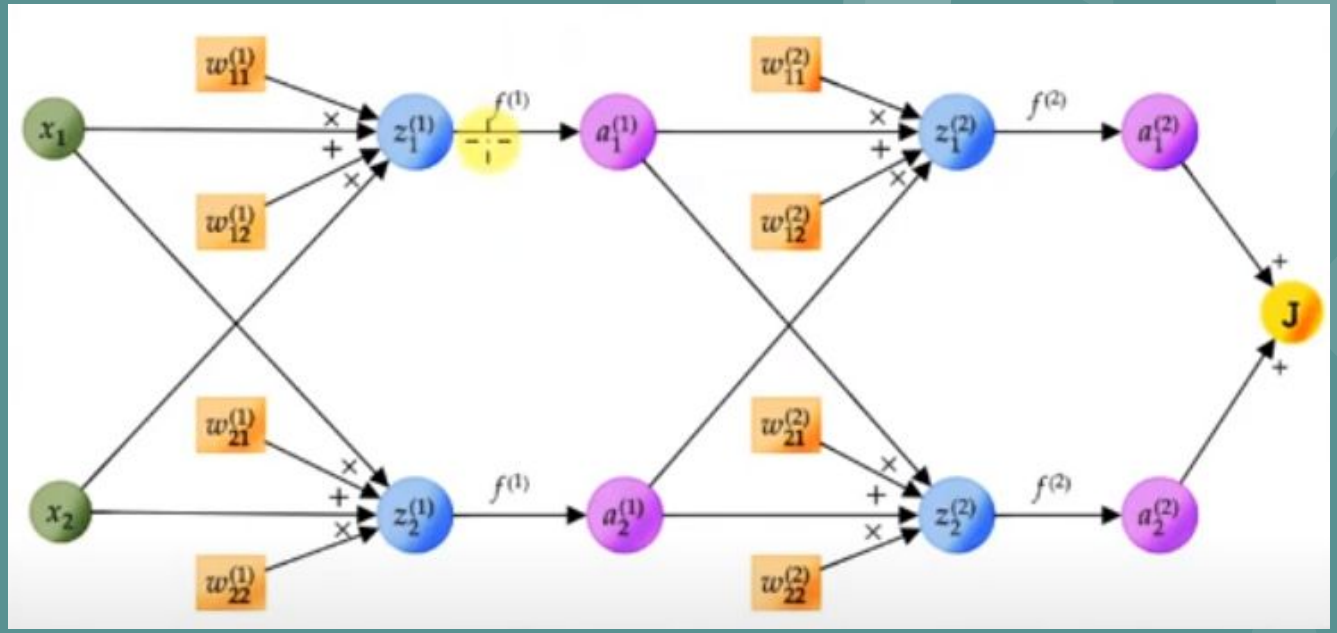


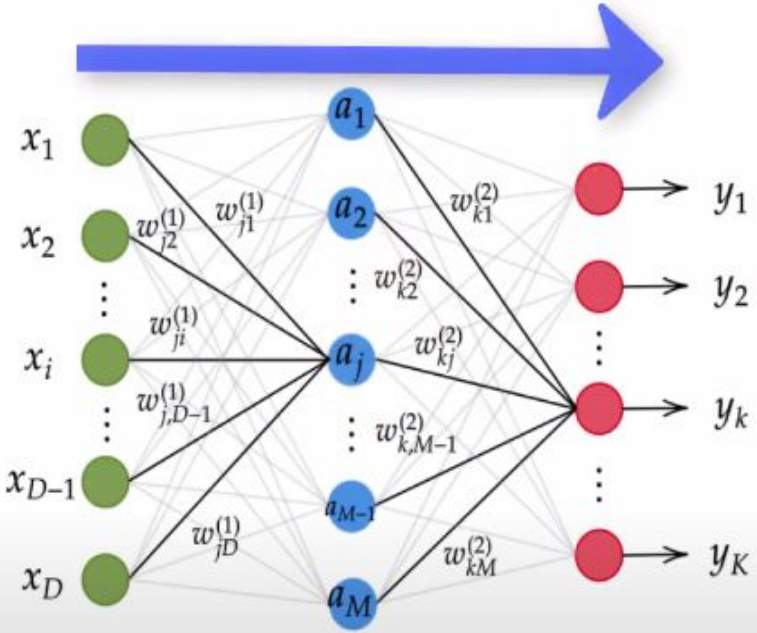
- ➔ Donde x1 y x2 son los datos de la capa de entrada y representan los datos de entrenamiento.
- ➔ Wij representa los pesos y el súper índice la capa a la que pertenecen.
- ➔ ak es el dato de salida para la capa k.
- ➔ yk son los datos de salida calculados por la red.

Forward propagation

- La red neuronal calcula y propaga las salidas desde la capa de entrada hasta la de salida.
- Es un proceso que se da capa por capa, hasta que se llega a una salida final.
- Pasos que se siguen en el forward propagation:
  - Se tienen los datos de la capa de entrada y sus respectivos pesos.
  - En la capa oculta, las entradas se combinan linealmente y pasan por una función de activación no lineal.
  - Finalmente, a la capa de salida llegan las obtenidas en la última capa oculta, aquí nuevamente se combinan de forma lineal y se pasa otra vez por una función de activación no lineal.

Grafo Computacional





1) $$z_j^{(1)} = w_{j1}^{(1)} x_1 + w_{j2}^{(1)} x_2 + \cdots + w_{jd}^{(1)} x_d = \sum_{i=1}^{d} w_{ji}^{(1)} x_i = \mathbf{w}_{j:}^{(1)\top} \mathbf{x}$$

2) $$a_j = f^{(1)}(z_j) = f^{(1)}\left(\sum_{i=1}^{d} w_{ji}^{(1)} x_i \right) = f^{(1)}\left(\mathbf{w}_{j:}^{(1)\top} \mathbf{x} \right)$$

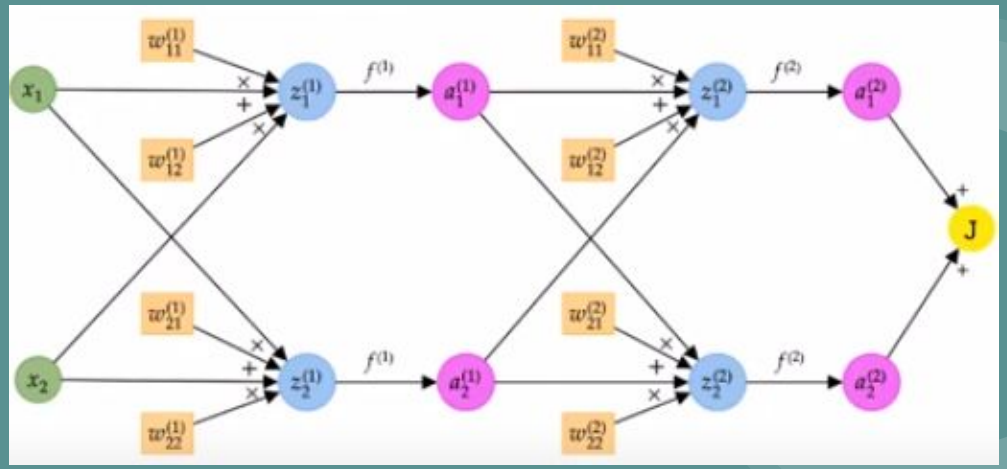
3) $$z_k^{(2)} = w_{k1}^{(2)} a_1 + w_{k2}^{(2)} a_2 + \cdots + w_{kM}^{(2)} a_M = \sum_{j=1}^{M} w_{kj}^{(2)} a_j = \mathbf{w}_{k:}^{(2)\top} \mathbf{a}$$

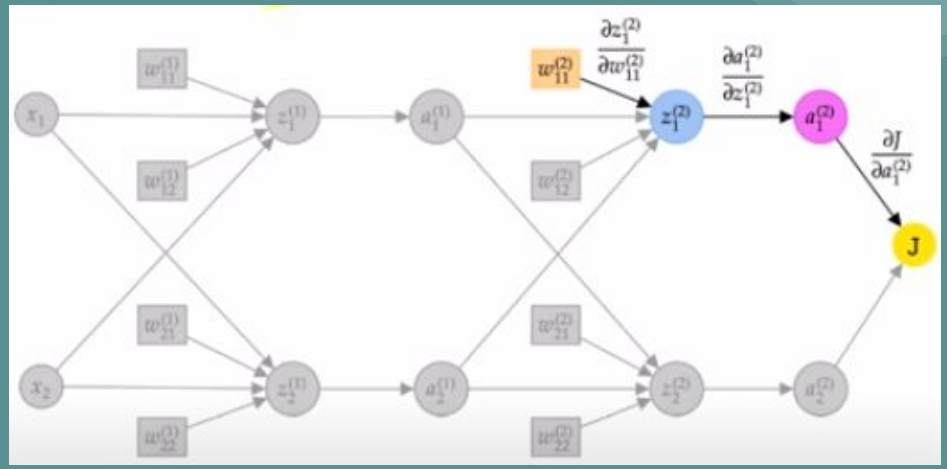
4) $$y_k = f^{(2)}\left(z_k^{(2)} \right) = f^{(2)}\left(\sum_{j=1}^{M} w_{kj}^{(2)} a_j \right) = f^{(2)}\left(\mathbf{w}_{k:}^{(2)T} \mathbf{a} \right)$$

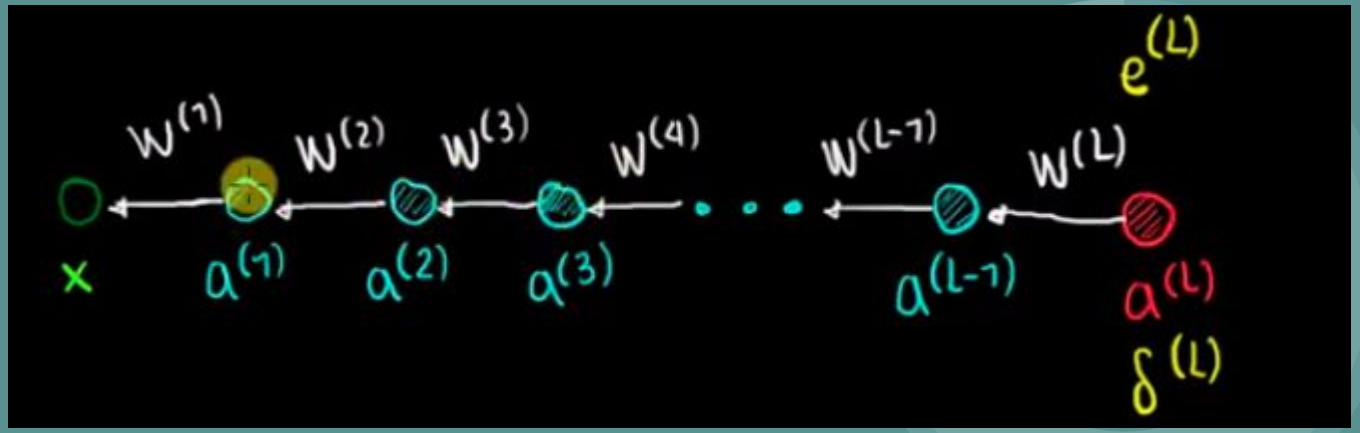
$$\begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ \vdots \\ z_M^{(1)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{1d}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{2d}^{(1)} \\ \vdots & \vdots & \ddots \vdots \\ w_{M1}^{(1)} & w_{M2}^{(1)} & w_{Md}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

Backpropagation

- También se conoce como retropropagación del error.
- Minimiza la diferencia entre las salidas calculadas y las deseadas.
- Aquí se necesita que la función de activación sea diferenciable, por lo que se usa el gradiente de la función sigmoide.
- Pasos que se siguen para el backpropagation:
  - Se realiza luego de que se haga el forward propagation.
  - Se calcula el error para mirar que tan bien está funcionando la red.
  - Se hace la retropropagación del error de atrás hacía adelante, esto mediante regla de la cadena.
  - Se hace un ajuste de los pesos haciendo uso del gradiente descendiente para minimizar el error.
  - Esto se realiza una y otra vez hasta que la red neuronal alcanza el número de épocas dadas.

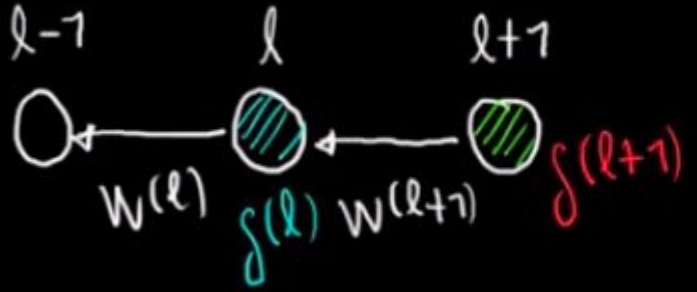

x_1
x_2
$w_{11}^{(1)}$
$w_{12}^{(1)}$
$w_{21}^{(1)}$
$w_{22}^{(1)}$
$z_1^{(1)}$
$z_2^{(1)}$
$f^{(1)}$
$a_1^{(1)}$
$a_2^{(1)}$
$w_{11}^{(2)}$
$w_{12}^{(2)}$
$w_{21}^{(2)}$
$w_{22}^{(2)}$
$z_1^{(2)}$
$z_2^{(2)}$
$f^{(2)}$
$a_1^{(2)}$
$a_2^{(2)}$
J


$\frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}}$
$\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}$
$\frac{\partial J}{\partial a_1^{(2)}}$
J



$$\begin{cases} e^{(L)} = d - a^{(L)} \\ \delta^{(L)} = f'(z^{(L)}) \odot e^{(L)} \end{cases}$$

Para cada $l \in \{L-1, L-2, \ldots, 3, 2, 1\}$



$$\begin{cases} e^{(l)} = W^{(l+1)^T} \delta^{(l+1)} \\ \delta^{(l)} = f'(z^{(l)}) \odot e^{(l)} \end{cases}$$

$$\begin{cases} \Delta W^{(l)} = \eta \left(\delta^{(l)} \otimes a^{(l-1)} \right) \\ W^{(l)} \leftarrow W^{(l)} + \Delta W^{(l)} \end{cases}$$

Código

- Mediante programación orientada a objetos se realiza una clase que tiene las siguientes características:
  - Se declaran los miembros: capaEntrada, capaOculta, capaSalida, pesosiniciales, pesosfinales.
  - En el constructor se inicializan los tamaños de las capas, mediante “this”.
  - Luego se libera memoria con el destructor.
  - Se definen las funciones de activación: sigmoide y sigmoide_derivada.

Código

- Se generan los pesos de manera aleatoria entre [-1,1].
- Se define la función de entrenamiento que recibe: X, y, eta, epocas y el tam_entrenamiento.
- Se realizan el forward y el backpropagation.
- Se actualizan los pesos.
- Se define la función predicción donde se reciben los parámetros: X y tamano_datos.
- Se realiza la predicción haciendo uso de la ya entrenada red neuronal.
- Se devuelve en arreglo con las predicciones.

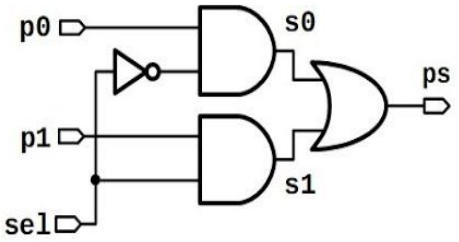
Ejemplo 1: Compuerta lógica OR

COMPUERTA OR		
A	B	Salida
0	0	0
0	1	1
1	0	1
1	1	1

Salidas red neuronal
0,00316093
0,99909
0,999111
0,999733

Capas de entrada:2
Capas ocultas: 5
Capas de salida:1

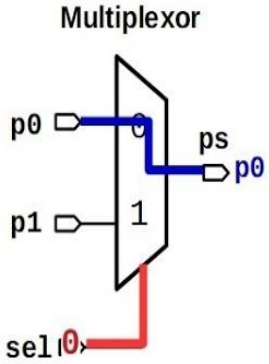
Ejemplo 2: Compuerta Multiplexor



sel	p1	p0	s1	s0	ps
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	0	1

Si sel = 0:
ps ← p0

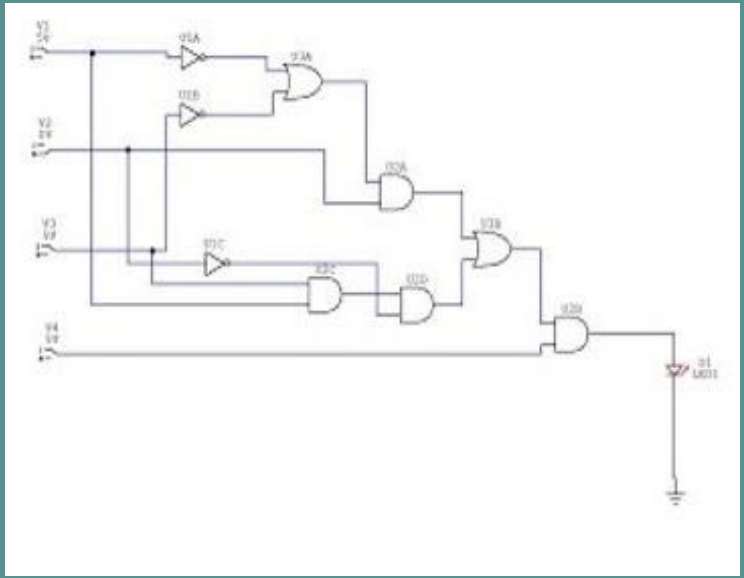
Si sel = 1:
ps ← p1



Salidas red neuronal
0,00243062
0,999121
0,000475942
0,999782
0,00104313
0,998928
0,996727
0,00259831

Capas de entrada:4.
Capas ocultas:7.
Capas de salida:1.

Ejemplo 3: Detector de números primos



Capas de entrada:4.
Capas ocultas:7.
Capas de salida:1.

N	A	B	C	D	RES	LED
0	0	0	0	0	0	OFF
1	0	0	0	1	0	OFF
2	0	0	1	0	1	ON
3	0	0	1	1	1	ON
4	0	1	0	0	0	OFF
5	0	1	0	1	1	ON
6	0	1	1	0	0	OFF
7	0	1	1	1	1	ON
8	1	0	0	0	0	OFF
9	1	0	0	1	0	OFF
10	1	0	1	0	0	OFF
11	1	0	1	1	1	ON
12	1	1	0	0	0	OFF
13	1	1	0	1	1	ON
14	1	1	1	0	0	OFF
15	1	1	1	1	0	OFF

N	Salidas red neuronal
0	0,003322737
1	0,000900177
2	0,99692
3	1
4	4,78302E-5
5	0,999008
6	0,0009305
7	0,99953
8	2,92296E-6
9	4,01014E-005
10	0,000627498
11	0,999484
12	1,00424E-5
13	0,997286
14	6,97058E-6
15	0,00144892