

- 
- Deep Learning
 - Fundamentos de la IA

DuocUC

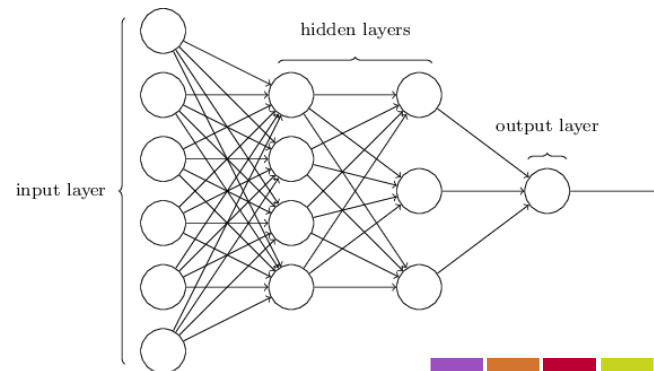


ESCUELA DE
INFORMÁTICA Y
TELECOMUNICACIONES





Redes Fully Connected Feed Forward



Próxima clase entregaré los detalles del encargo!



EVALUACIONES

F
e
c
h
a
s

P
o
n
d
e
r
a
c
i
ó
n

Unidad 1
(prueba)

15-04-23

10%

Unidad 2 (Encargo +
Presentación)

06-05-23

40%

Unidad 2 (Encargo
+ Presentación)

03-06-23

25%

Unidad 4
(Encargo +
Presentación)

01-07-23

25%

70%

Examen transversal (Encargo + Presentación) 08-07-23

30%

Calendario

MARZO

SM	LU	MA	MI	JU	VI	SA	DO
09			1	2	3	4	5
10	6	7	8	9	10	11	12
11	13	14	15	16	17	18	19
12	20	21	22	23	24	25	26
13	27	28	29	30	31		

ABRIL

SM	LU	MA	MI	JU	VI	SA	DO
13						1	2
14	3	4	5	6	7		9
15	10	11	12	13	14	15	16
16	17	18	19	20	21	22	23
17	24	25	26	27	28		30

MAYO

SM	LU	MA	MI	JU	VI	SA	DO
18	1	2	3	4	5	6	7
19	8	9	10	11	12	13	14
20	15	16	17	18	19	20	21
21	22	23	24	25	26	27	28
22	29	30	31				

JUNIO

SM	LU	MA	MI	JU	VI	SA	DO
22				1	2	3	4
23	5	6	7	8	9	10	11
24	12	13	14	15	16	17	18
25	19	20	21	22	23	24	25
26	26	27	28	29	30		

JULIO

SM	LU	MA	MI	JU	VI	SA	DO
26						1	2
27	3	4	5	6	7	8	9
28	10	11	12	13	14	15	16
29	17	18	19	20	21	22	23
30	24	25	26	27	28	29	30
31	31						

Resumen Inicial

El perceptrón como única neurona tiene varias limitaciones y en la medida que usamos más neuronas, se genera una mayor **capacidad** para resolver problemas cada vez más complejos.

Las redes neuronales tienen una capa de entrada (X) que en la práctica corresponde a un vector que contiene todos los ítems del ejemplo o datos.

Ejemplo: Si es un texto que tiene 100 caracteres, el vector será de largo 100 y cada x_i contendrá una letra del texto.

Las redes neuronales **SOLO** reciben números como entrada.

La red del ejemplo tiene 1 capa “ h ” (hidden), llamada capa oculta, en la cual hay varios perceptrones (MLP – Multi Layered Perceptron) para darle una mayor capacidad de resolver problemas complejos.

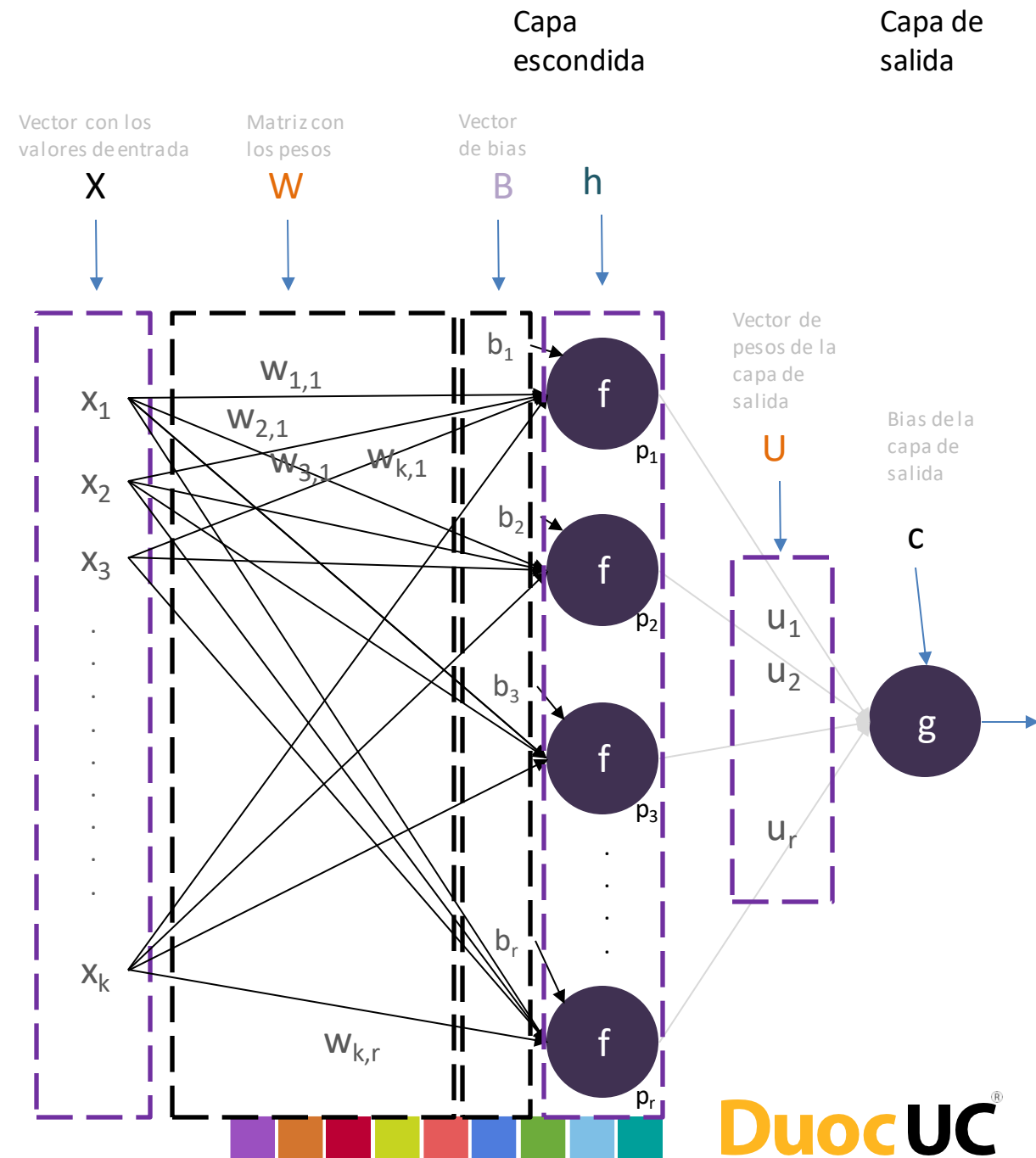
No hay una fórmula para poder determinar el número de neuronas por capa oculta, sino que la experiencia define cuántos utilizar.

La función de activación utilizada en todas las neuronas de una capa oculta, debe ser la misma y que la más utilizada hoy es **ReLU**. Razones: Es rápida, su derivada es simple de calcular, y dado que la derivada es siempre 0 o 1, evita el desvanecimiento del gradiente, es decir, que la red no aprenda ya que la variación en las pendientes (gradientes) es muy poca, como pasa con sigmoid o Tanh en la medida que se alejan del eje y .

Existe la función de salida “ g ”, que no es el mismo concepto que una función de activación. Puede ser cualquiera, pero hay funciones recomendadas para cada caso. La función de salida, calcula finalmente la salida calculada de la red “ \hat{y} ”, que puede ser un escalar o un vector, dependiendo de lo que estamos haciendo con la red. Esta salida calculada se compara con la salida esperada.

Los pesos (w) y bias (b) constituyen los parámetros de la red y que sus valores (números reales) son **inicializados al azar** al comenzar el entrenamiento.

Cada perceptrón p_i , debe recibir un peso por cada entrada x_i y también un bias b_i por cada perceptrón.



Redes Fully Connected – Feed Forward

Feed Forward: la información fluye hacia adelante (Desde la primera capa hacia la última)


Fully Connected: las “neuronas” están inter conectadas todas con todas entre una capa i y su sucesora $i+1$

Desde ahora **Redes neuronales profundas (Deep learning)** ya que pueden tener muchas capas ocultas, según la gráfica

Al igual que para determinar la cantidad de neuronas por capa no existe una fórmula; tampoco existen una para definir la cantidad de capas de una red neuronal profunda

Al aumentar la cantidad de neuronas por capa y/o el número de capas de la red, aumentan drásticamente la cantidad de parámetros [pesos (w) y bias (b)]. De la misma manera aumenta el tiempo de duración del “entrenamiento” de la red

Redes muy famosas tienen billones de parámetros (pesos y bias)

 **NVIDIA. DEVELOPER**

[HOME](#) [BLOG](#) [FORUMS](#) [DOCS](#) [DOWNLOADS](#) [TRAINING](#)

Get online training, developer insights, and access to experts at GTC 2022.

TECHNICAL BLOG

SUBSCRIBE

NEWS

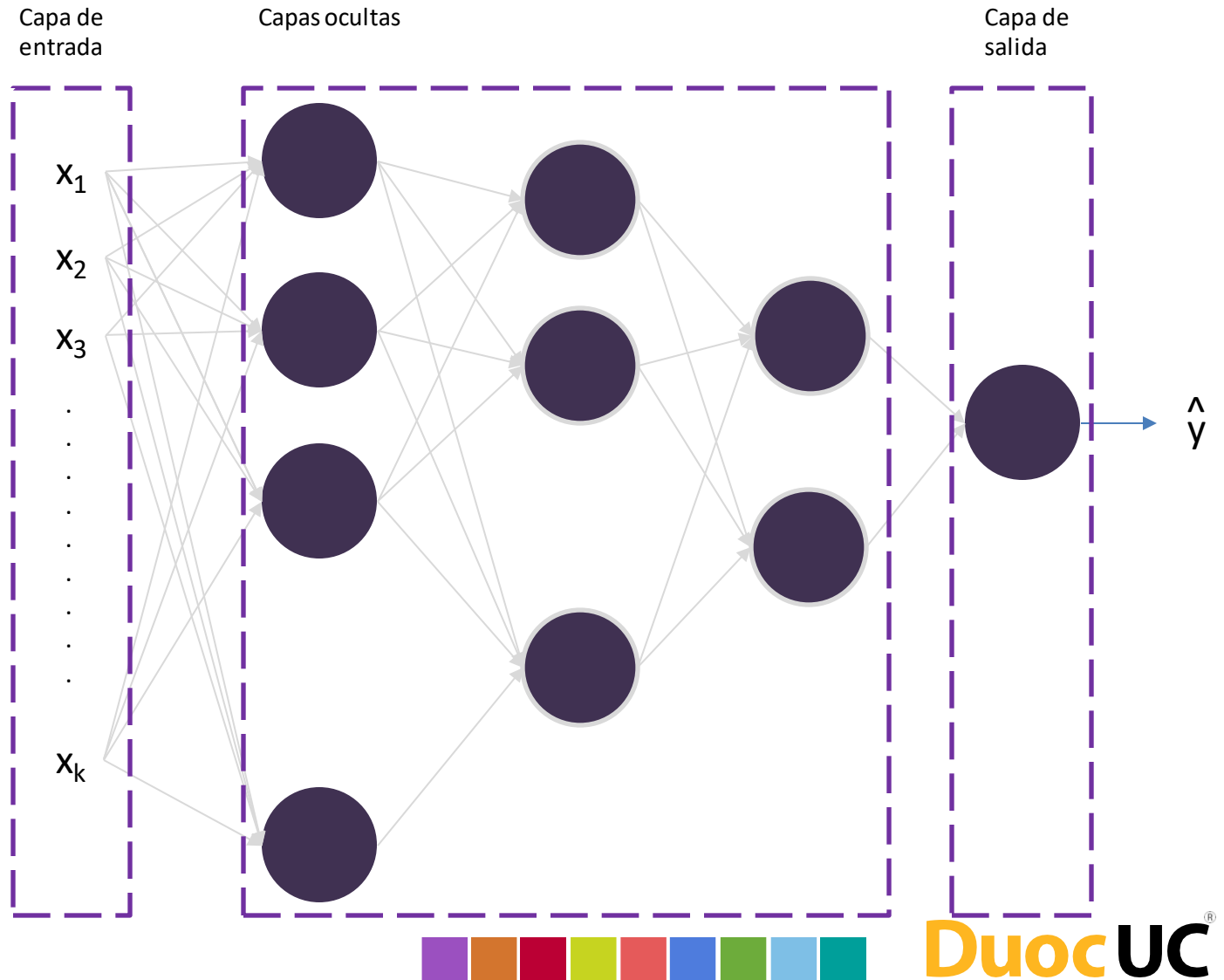
Apr 12, 2021

Announcing Megatron for Training Trillion Parameter Models & NVIDIA Riva Availability

Discuss (0) Share 0 Like

Tags: featured, Machine Learning & Artificial Intelligence, News, Riva

Esquema de alto nivel de una red feed forward de 3 capas



Redes Fully Connected – Feed Forward

Generalizando la representación de la red feed forward

La entrada se denomina capa de entrada y se representa con una X

La matriz de pesos que se produce entre capas (ya sea la capa de entrada y la primera capa oculta o, entre capas ocultas), se denomina con la W mayúscula y con un superíndice “ i ” que indica el número de la capa a la cual pertenece.

Cada capa tiene una función de activación “ f ” con su respectivo superíndice indicando el número de la capa a la cual se refiere

Cada capa oculta “ h ”, su vez tiene un vector de sesgo o bias, que representamos con la letra b y también con un superíndice para saber a que capa pertenece

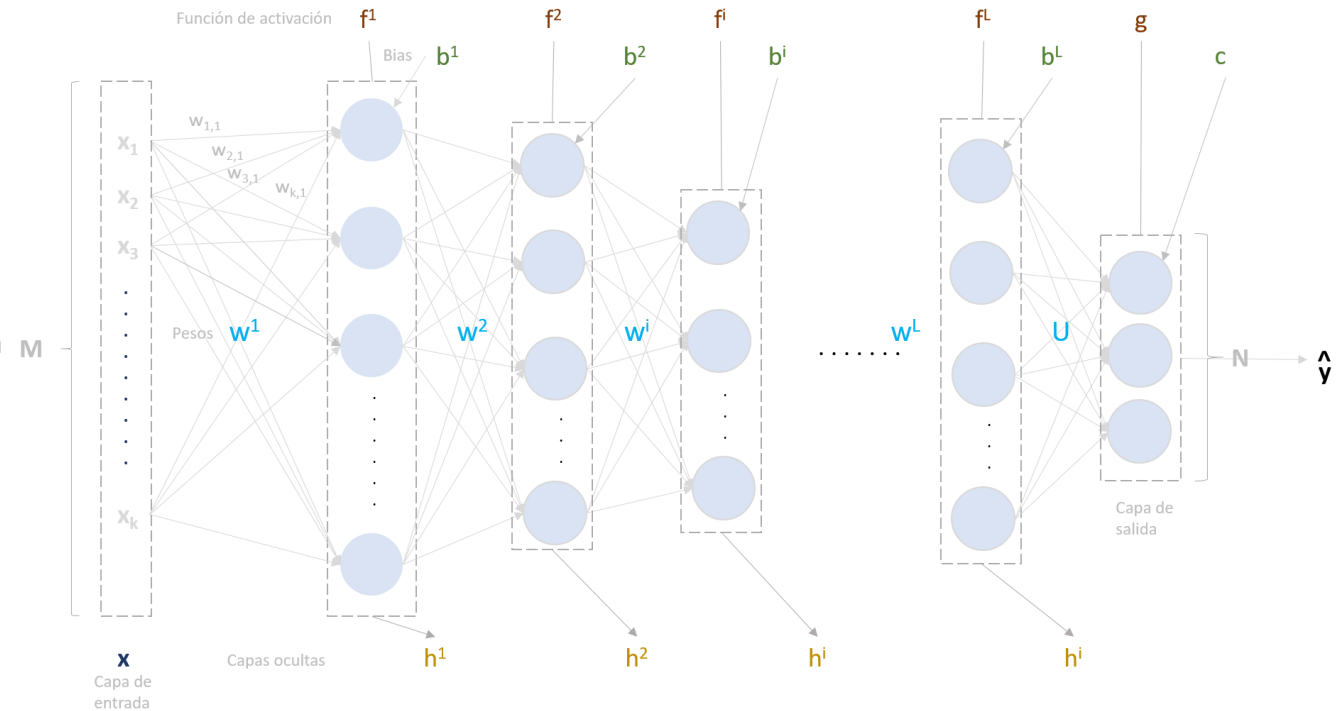
Esta cantidad de capas escondidas se denota con la letra “ L ”.

U es la matriz que tiene los parámetros de la última capa, previa a la capa de salida.

La capa de salida tiene una función de salida que la definimos con la letra “ g ” y un bias cuya notación es la letra “ c ” minúscula.

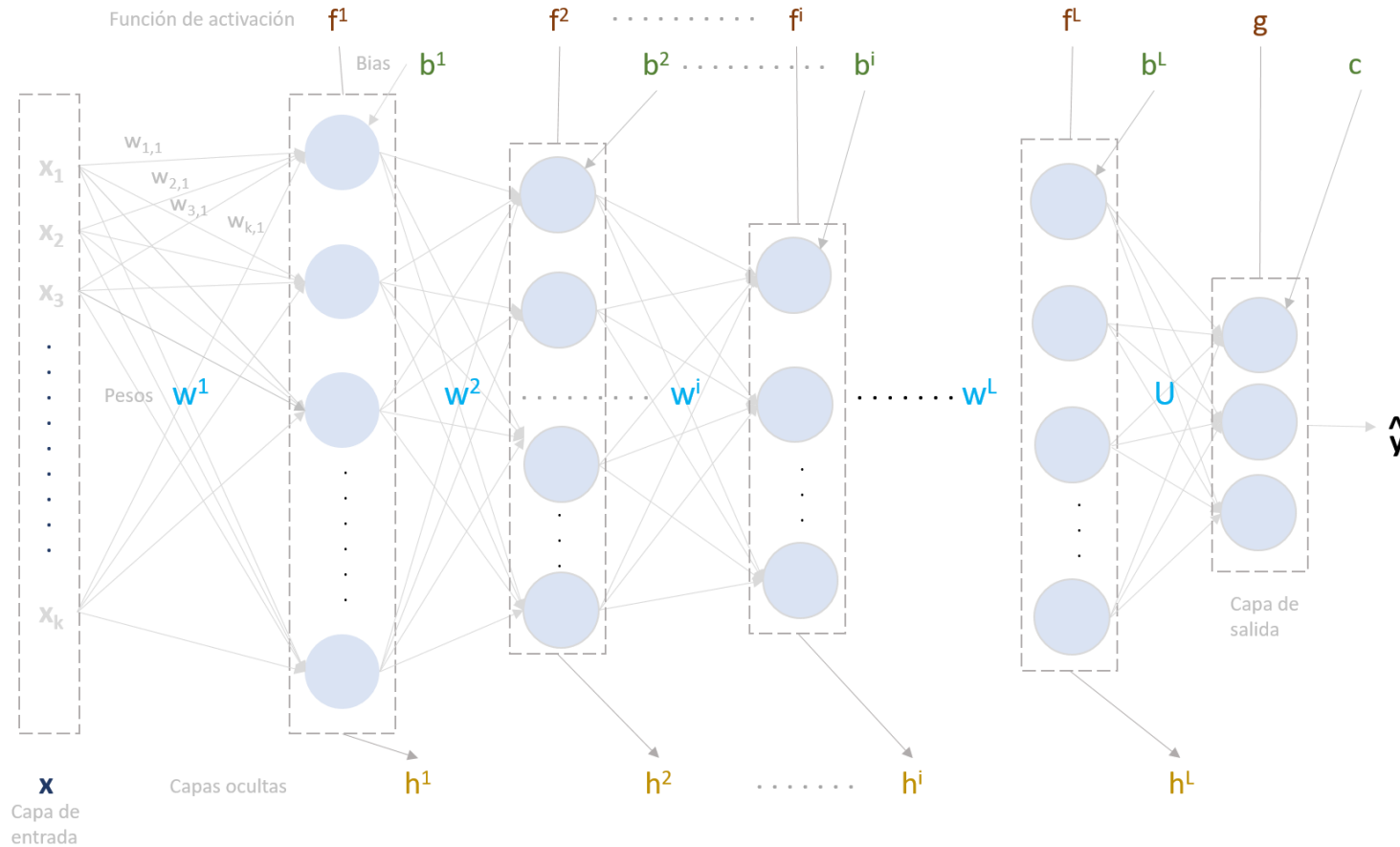
El tamaño de la capa de es “ M ” y el tamaño de la capa de salida, “ N ” (cantidad de neuronas)

La cantidad de parámetros que posee la red se usa, por ejemplo, para determinar el hardware necesario que se necesita para entrenar las redes neuronales profundas.



Redes Fully Connected – Feed Forward

Generalizando la representación de la red feed forward



Salida de Capa Oculta

$$h^{(i)} = f^{(i)}(h^{(i-1)} * W^{(i)} + b^{(i)})$$

$$\hat{y} = g(h^{(L)} * U + c)$$

Salida de Capa de Salida

Función de la capa de Salida

La función que se utilizaba en la capa de salida es, o puede ser diferente, a las usadas como función de activación de las capas ocultas.

La función de salida depende del problema que quiero resolver (predicción o clasificación)

Para clasificación, una de las funciones más utilizadas es **Softmax**

Al clasificar, en general, la salida es una distribución de probabilidades sobre las clases

Clases es “cuántas posibles salidas tenemos”, por ejemplo, al clasificando estados de ánimos : pena, alegría, depresión, ansiedad, se dice que hay 4 clases.

Para el ejemplo anterior, la salida de la red debería ser un vector de cuatro neuronas, donde cada una de ellas contiene un número real al cual nosotros le damos una interpretación y representa la probabilidad de que una de ellas sea la “respuesta o clasificación” correcta

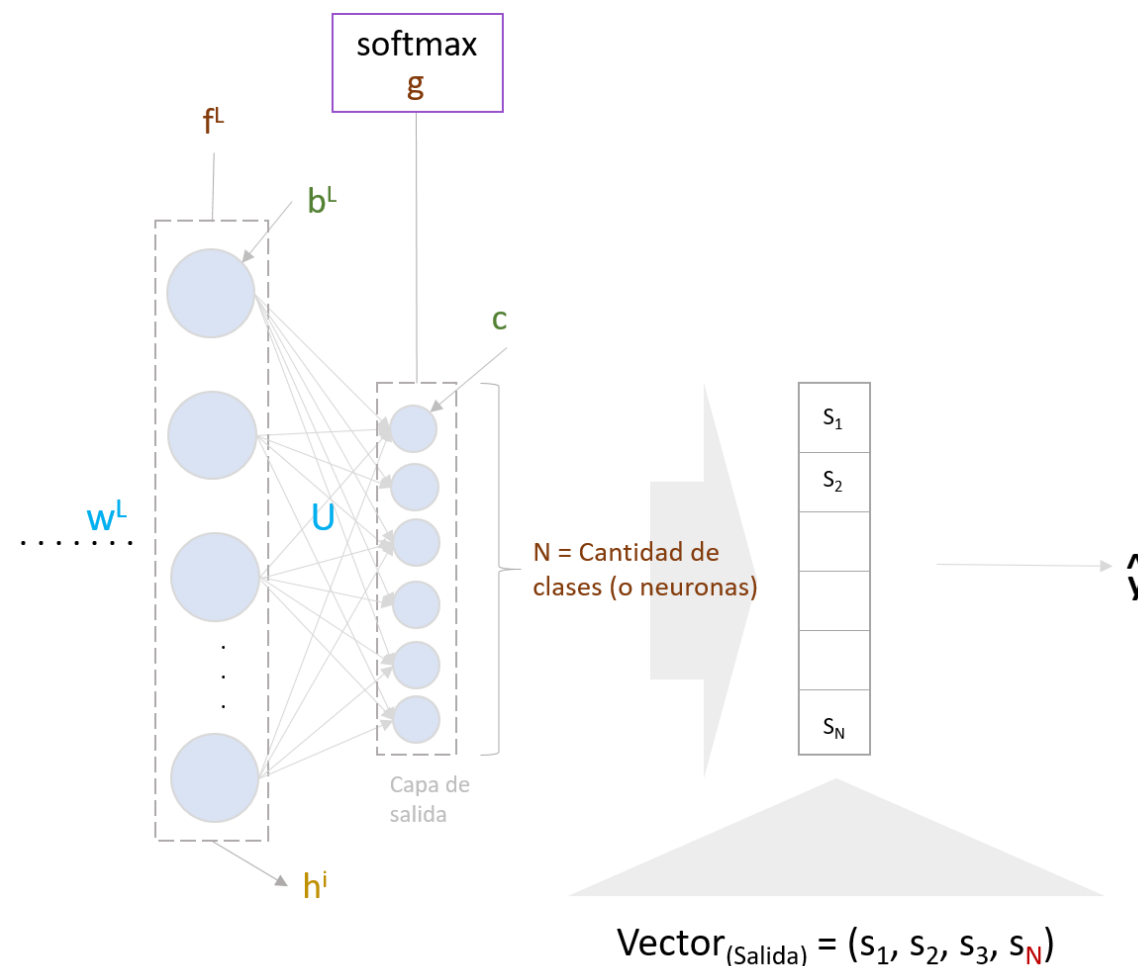
En otras palabras, al aplicar softmax a la capa de salida, en una de las 4 posiciones del vector de salida, hay un 0,90 en la clase “alegría”, significa que hay una probabilidad de un 90% que el estado de ánimo sea “alegría”, entonces una salida puede tener probabilidad de ser de todas las clases, pero en menor medida

Para esto, es necesario que la función de salida permita hacer dos cosas:

- Asegurar que todos los números queden entre 0 y 1
- Asegurar que la suma de todos los números sea 1

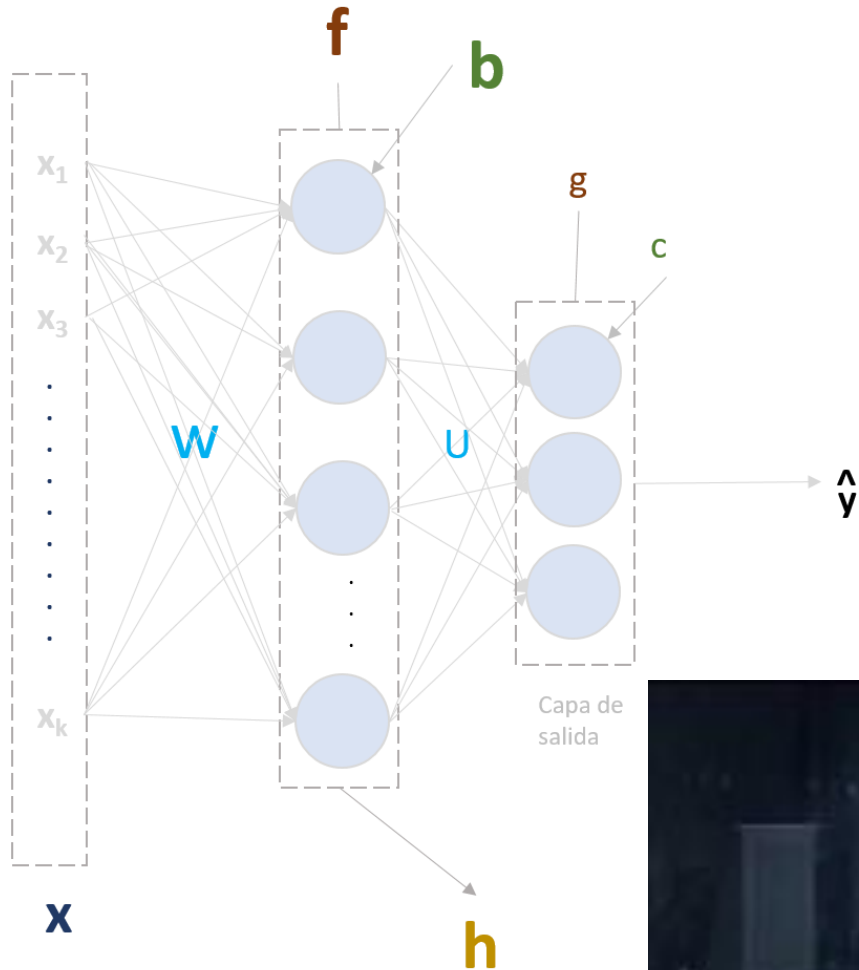
Con esto, se puede interpretar cada uno de los valores del vector de salida como una distribución de probabilidad

Softmax, como función de salida, asegura que el vector que se generó, se puede interpretar como una distribución de probabilidades.



Aspectos de implementación

$$h = f(X*W+b)$$



- Multiplicar matrices es “caro” computacionalmente (en este caso el vector x y la matriz de los pesos)
- Una de las ventajas hoy, es usar una librería (y hardware especializado (GPU, TPU)) que permita hacer las operaciones con matrices de manera paralela



- En la práctica, se pasan o ingresan a la red un conjunto de ejemplos, en lugar de pasar de 1 ejemplo a la vez de manera secuencial.
- Entonces, X ya no sería un vector de $(k,1)$, sino que sería una matriz de (k,n)
- Esto es lo que permite que hoy las redes neuronales funcionen dado que existe el HW que permite este cálculo en paralelo
- La cantidad de ejemplos a pasar en paralelo, dependerá de la capacidad de la GPU

Entrenamiento

Entrenar significa que la red encuentre los valores correctos para todos los pesos y bias (para todos sus parámetros)

Primero, se deben inicializar mediante algún método que veremos más adelante.

Para realizar el entrenamiento, se necesita calcular estos valores, los que son números reales, muy pequeños (centrados en el cero).

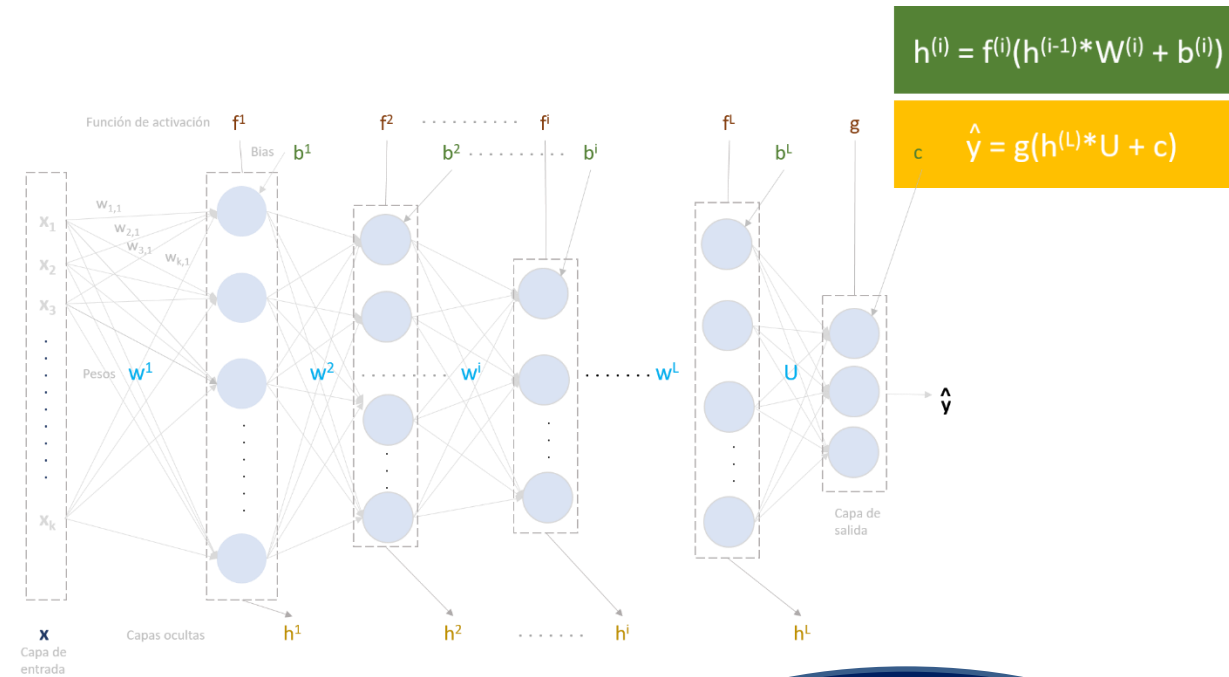
Los parámetros (pesos y bias) son calculados por algoritmos que se revisarán en detalle más adelante.

La red itera, entre la primera capa oculta y la capa de salida hasta encontrar la mejor combinación de pesos y bias que resuelva el problema.

El proceso de avanzar desde la primera capa hasta el resultado esperado (y) se le conoce como forward(x).

Hay algunos procesos intermedios que revisaremos más adelante.

Para poder entrenar la red, en general, se necesitan son datos y ojalá miles de datos, inicializar los parámetros e iterar hasta que se encuentre la solución que se parezca a la óptima para el problema a resolver.



Entrenar una red neuronal es encontrar los valores "correctos" para W, b, U y c

Entrenamiento

Lo primero que se necesita para entrenar una red, son datos y en general como materia prima, el dato (entrada / muestra) y la salida (respuesta) que se espera obtener.

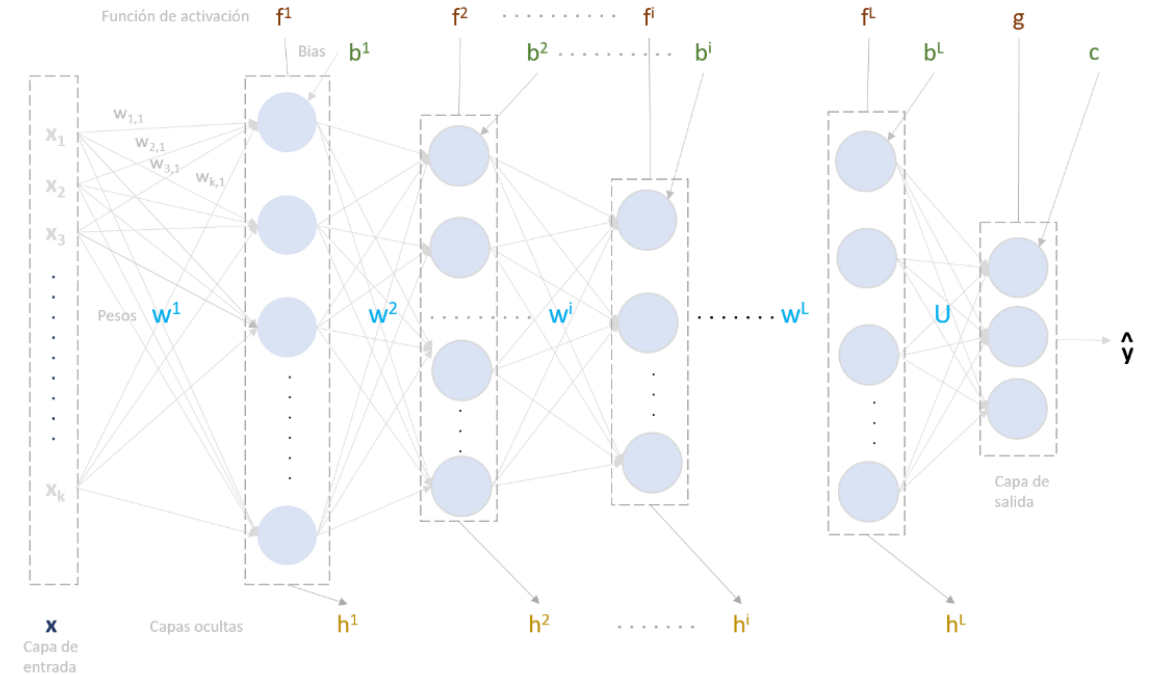
$$D = \{ (x_1, y_1), (x_2, y_2) \dots (x_i, y_i) \dots (x_n, y_n) \}$$

Donde:

D : Es el conjunto de datos o ejemplos a utilizar para entrenar la red

x_i : Es un ejemplo o dato a ingresar a la red

y_i : Es el resultado que esperamos calcule la red



Datos de entrada

Resultado esperado

$x_i \longrightarrow y_i$

\hat{y}_i

Resultado que calculó la red

Entrenamiento – Función Error (Loss)

y_i representa el valor que espero la red me calcule

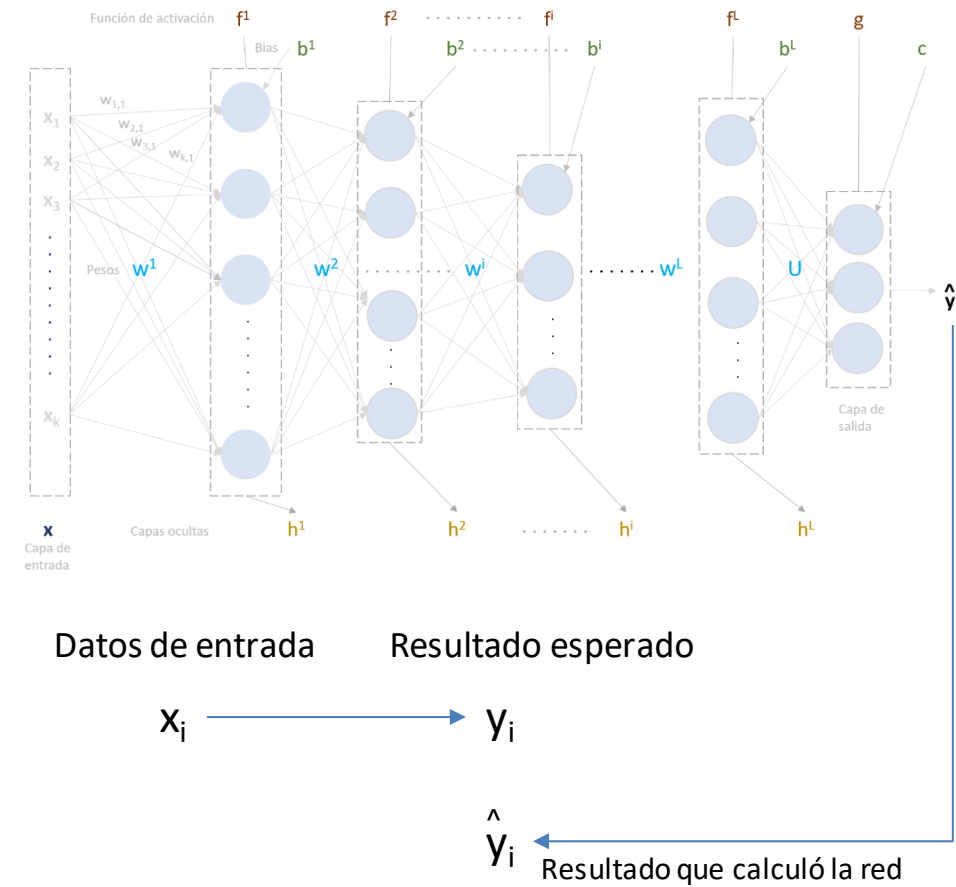
\hat{y} es el resultado que calculó la red durante el entrenamiento

Para poder determinar si la red está calculando de manera correcta, debe existir alguna medida que permita calcular la diferencia entre lo esperado y lo calculado, es decir, una medida que nos indique cuánto se está equivocando la red

Este proceso se hace a través de la “función de error” [más adelante veremos cómo se implementa]

La función de error (\mathcal{L}) representa la diferencia entre la salida esperada (y) y la salida calculada (\hat{y})

$$\mathcal{L} = (y_i ; \hat{y}_i)$$



Función Error (Loss)

El resultado de esta función de error (\mathcal{L}) es un número real, un escalar

Indica el error o diferencia para un ejemplo, sin embargo, esto no es suficiente, lo esperado es que la red se equivoque en promedio muy poco en todos los ejemplos que ingresados

Lo que hará la red entonces, es promediar todos los errores que comete por cada ejemplo

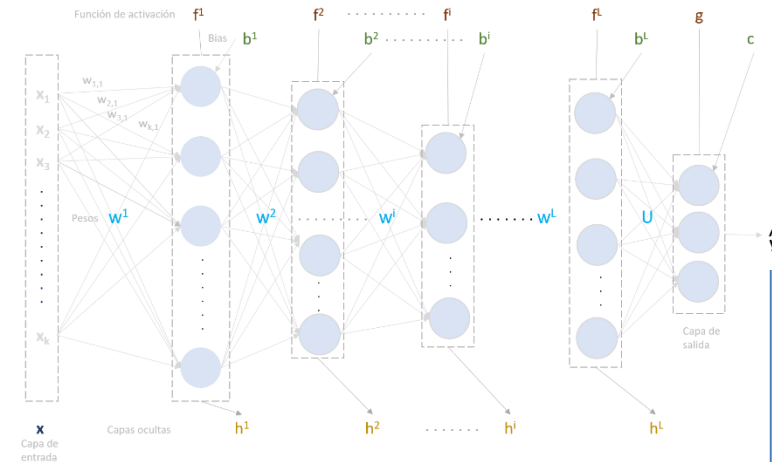
A este error promedio de todos los errores cometidos en cada ejemplo, se le llama en **Deep learning “pérdida” o “loss”**

Esta función, en otras palabras, indica qué tanto se está equivocando la red.

Este promedio es un número real que, aspiracionalmente, debe ser muy chico, ojalá cero

$$\mathcal{L} = (y_i ; \hat{y}_i)$$

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n \text{error}(\hat{y}^{(i)} ; y^{(i)})$$



Datos de entrada

Resultado esperado

x_i

y_i

\hat{y}_i

Resultado que calculó la red

Función Error (Loss)

Esta función de error depende de los parámetros, de todos los parámetros de la red, es decir, theta es igual a todos los W^i , b^i , U y c

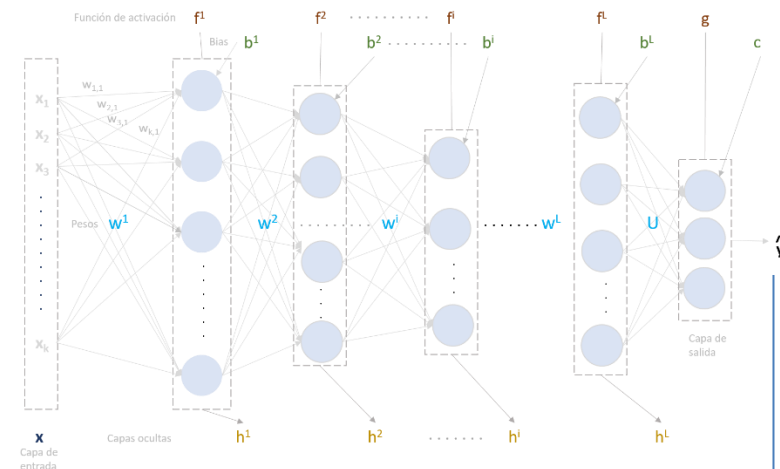
$$\Theta = (W^i, b^i, U, c)$$

Asumiendo que no tenemos más ejemplos que agregar al set de datos, **la función de error depende exclusivamente de los parámetros** (pesos y bias)

Al existir esta dependencia, para que la función de error cambie e idealmente tienda a cero, deben cambiar o ajustarse los parámetros de la red

Y para finalizar el concepto de entrenamiento, en términos matemáticos, **entrenar significa encontrar el theta (Θ) que minimiza la función de error (\mathcal{L})**

$$\Theta = (W^i, b^i, U, c)$$



Datos de entrada Resultado esperado

$$x_i \longrightarrow y_i$$

\hat{y}_i Resultado que calculó la red

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n \text{error}(\hat{y}^{(i)} - y^{(i)})$$