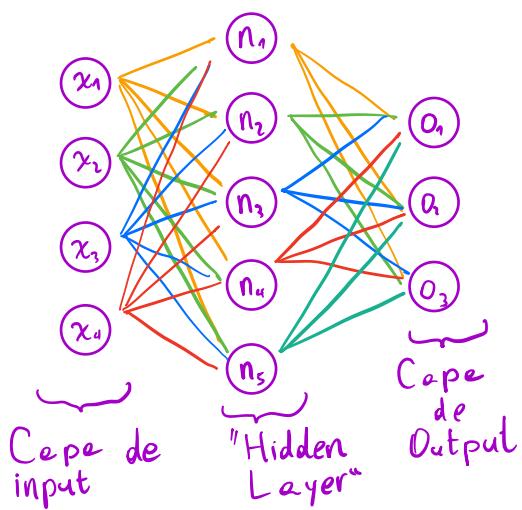


Redes Neuronales

Las Redes Neuronales son la herramienta más utilizada en el área de Machine Learning. La idea es (a grandes rasgos) replicar la forma de "aprender" de los humanos. Vamos a partir por un ejemplo. Supongamos el dataset Iris de 4 features $\{x_1, x_2, x_3, x_4\}$, en el que queremos clasificarlas en 3 tipos $\{1, 2, 3\}$. Una red neuronal se ve de la siguiente forma:

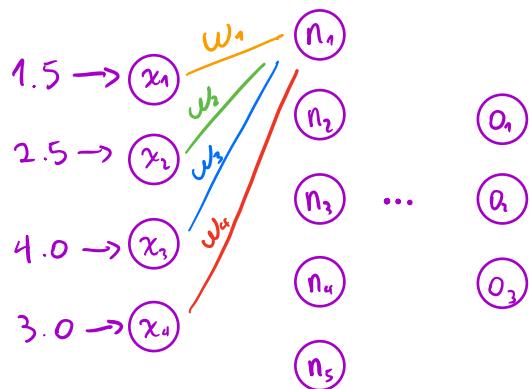


En donde tenemos una capa de input, una "hidden layer" y una capa de output. Hay que notar que la "hidden layer" se compone de las neuronas $\{n_1, \dots, n_5\}$ y este capa es una "fully connected" porque cada neurona se conecta a todos los nodos de la capa anterior y posterior.

Ahora, ¿Cómo funciona este red? Supongamos que

queremos clasificar una flor con los siguientes valores:

$$\{x_1: 1.5, x_2: 2.5, x_3: 4.0, x_4: 3.0\}$$



En la red, cada conexión entre capas tiene un peso. Aquí estamos mostrando los pesos de las conexiones entre la capa de input y la neurona n_1 . Así, n_1 guardaría el valor:

$$V_{n_1} = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + b_1$$

$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$

$$1.5 \quad 2.5 \quad 4.0 \quad 3.0$$

Notamos que, además de los valores de las conexiones la neurona guarda un valor b_1 , que llamemos "bias". Así, debido a la conexión de la capa de input y la "hidden layer" tenemos $4 \cdot 5 = 20$ pesos y 5 bias.

$\swarrow \quad \searrow$
 4 inputs 5 neuronas.

Así, si llamamos w_{ij} el peso de la conexión entre la neurona i y el input j podemos definir la matriz M_i :

$$M_i = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \\ w_{51} & w_{52} & w_{53} & w_{54} \end{bmatrix}$$

Y si consideramos el bias de cada neurona i como b_i y el input j como x_j , tenemos que:

$$\begin{bmatrix} v_{n_1} \\ v_{n_2} \\ v_{n_3} \\ v_{n_4} \\ v_{n_5} \end{bmatrix} = M_i \vec{x} + \vec{b}_i \quad \text{con} \quad \vec{b}_i = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix} \quad \text{y} \quad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

De la misma forma, podemos definir M_2 como la matriz de 3 filas y 5 columnas con los pesos de las conexiones entre la "hidden layer" y la capa de output, así:

$$\begin{bmatrix} o_1 \\ o_2 \\ o_3 \end{bmatrix} = M_2 \cdot \begin{bmatrix} v_{n_1} \\ v_{n_2} \\ v_{n_3} \\ v_{n_4} \\ v_{n_5} \end{bmatrix} + \vec{b}_2 \quad \text{con} \quad \vec{b}_2 \quad \text{el vector de bias para estas conexiones.}$$

Y la instancia nueva es clasificada con la clase representada por el ok más grande.

Así, para este caso tenemos que aprender:

$$4 \cdot 5 + 5 \cdot 3 = 20 + 15 = 35 \text{ pesos}$$

$$5 + 3 = 8 \text{ bias}$$

Y una vez aprendidos estos valores, para clasificar solo debemos multiplicar matrices y sumar vectores.

Antes de aprender cómo se entrena las redes, vamos a presentar unas ecotaciones.

Sobre las distintas capas

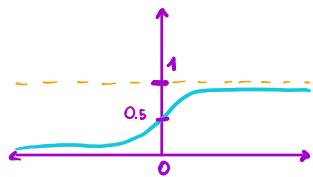
En general, un predictor basado en redes neuronales tiene varias "hidden layers". Además cada una de estas capas tiene asociada una función de activación $f(\cdot)$. Así, cuando calculamos el valor calculado en una neurona n , el resultado lo pasamos por la función de activación. En el ejemplo:

$$V_{n_1} = f(w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + b_1)$$

↓ ↓ ↓ ↓
1.5 2.5 4.0 3.0

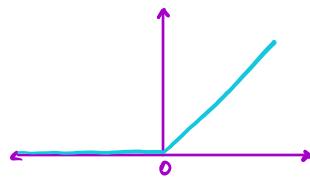
Algunas funciones de activación ampliamente usadas son:

Sigmoide:



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

ReLU:

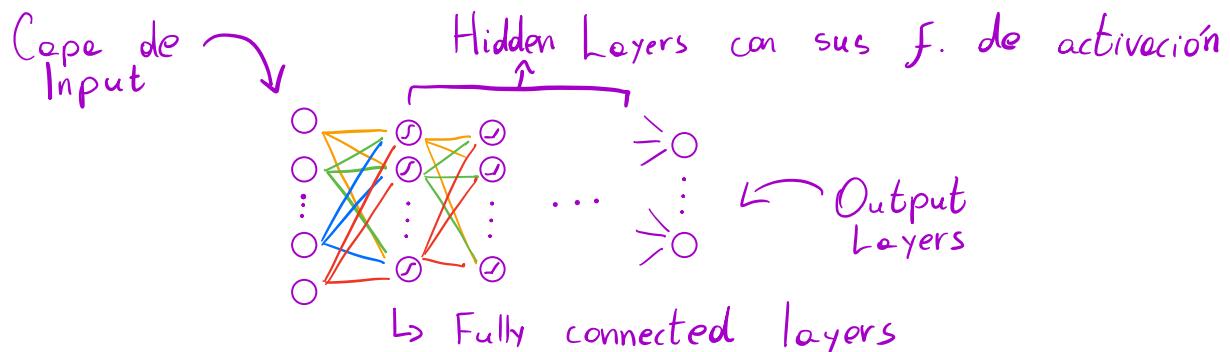


$$\text{ReLU}(x) = \max(0, x)$$

Estas funciones son importantes pues son las que nos permiten hacer **clasificación no lineal**.

También la capa de output puede estar asociada a una función de activación Softmax para que el vector de output sea representado como un vector de probabilidades sobre las clases.

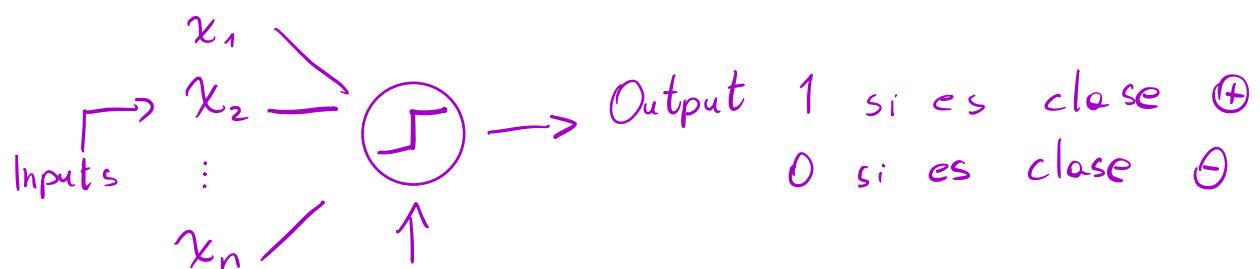
Así, una arquitectura de varias "hidden layers" es conocida como una **"Deep Neural Network"**



A esta arquitectura, en que cada hidden layer esté completamente conectada a la capa anterior y a la siguiente (fully connected) se le conoce como **Multi Layer Perceptron**.

El perceptrón

El perceptrón es una técnica de clasificación inventada en 1957, que es la arquitectura más sencilla de redes neuronales:



Calcula
 $\Sigma(\vec{w} \cdot \vec{x} + b)$

$$\Sigma(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$$

Con \vec{w} vector de pesos
 \vec{x} vector de input

Como lo vemos, es una única neurona que calcula un valor como lo vimos antes y clasifica según el signo. Muy parecido a SVM lineal o Regresión Logística.

Finalmente, aquí podemos entender que la idea de las redes neuronales es similar a como aprende un humano: una neurona se activa o no dependiendo del input, y los pesos \vec{w} los aprendió de ejemplos pasados. Así, una red multiplica ve fortaleciendo (aumenta el peso) o debilitando (disminuye el peso) las conexiones entre neuronas al aprender. Luego al ver una instancia desconocida, se van activando neuronas que producen activaciones en capas siguientes y así hasta entregar un output.

Ahora, tenemos que entender cómo entrenamos una red neuronal. Esto es, cómo encontramos los valores de los pesos y los bias.

Backpropagation

La técnica de Backpropagation nos sirve para ejecutar Gradient Descent de forma eficiente gracias a la regla de la cadena. No veremos los detalles técnicos, pero este es lo idea:

- Al inicio, inicializamos de manera aleatoria todos los pesos y bies.

- Luego le mostramos un ejemplo etiquetado. Supongamos que es de clase 2 y el resultado fue el siguiente:

$$O_1 = 0.5$$

$$O_2 = 0.35$$

$$O_3 = 0.15$$

¡Pero nosotros queríamos este!

$$O_1 = 0$$

$$O_2 = 1$$

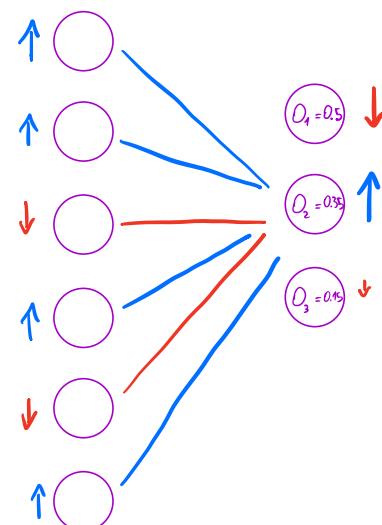
$$O_3 = 0$$

(O_{j0} , como en regresión softmax, en general optimizamos la función objetivo (cross-entropy))

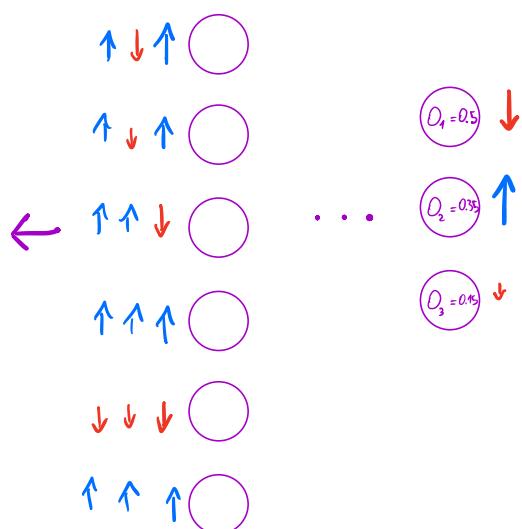
Así, le decimos a la capa de output que queremos que O_1 decrezca mucho, O_3 decrezca "poco" y O_2 incremente mucho:

Así, pero que O_2 tiene un valor mayor, necesitamos:

- Incrementar el bias en O_2
- Incrementar los pesos de las conexiones a los neuronas con mayor valor numérico (i.e. neuronas "más activas")
- Pedirle a las neuronas conectadas e pesos negativos que se activen menos y a las con pesos positivos se activen más (proporcional al peso!)



Luego, juntamos los "deseos" de los otros neuronas de Output:



Y luego agregamos esos deseos y los pasamos a la hidden layer anterior

Después de esto, tendremos un valor numérico en el que queremos variar cada peso y bias de la red solo para este ejemplo. Luego, le deberíamos mostrar cada ejemplo para agregar los valores numéricos que cada ejemplo pide cambiar y así obtenemos el gradiente. En general usamos mini-batch gradient descent. Esto es, considerar una porción del dataset en cada iteración de Gradient Descent. En el entrenamiento, la red verá el dataset entero varias veces. Cada una de estas veces es una **época** (**epoch**). Los datos se reordenan aleatoriamente

después de cada época. Y así, podemos entrenar nuestra red.

Redes Neuronales y Regresión

Para hacer regresión con una red neuronal simplemente la capa de output tendrá una neurona que calculará el valor a predecir. En general, esa neurona no usará ninguna función de activación.