Clase 17: Redes neuronales (parte 1)

MDS7104 Aprendizaje de Máquinas

Felipe Tobar

Iniciativa de Datos e Inteligencia Artificial
Universidad de Chile

31 mayo 2024

UNIVERSIDAD
DE CHILE

Redes Neuronales - Introducción

Una *Red Neuronal* es un modelo computacional basado en la conexión de múltiples unidades (neuronas) cuyo objetivo* es aproximar una función $x \mapsto y = f(x)$.

Su principal ventaja radica en la posibilidad de ser entrenarla con datos *crudos*, es decir, de forma agnóstica con respecto de las características relevantes de los datos, con lo que la red *aprende* las características.

Los modelos escenciales de redes neuronales se conocen como **feedforward neural networks**, o **multilayer perceptrons** (MLPs), Una red feedforward define un mapa $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$, donde los parámetros $\boldsymbol{\theta}$ se aprenden para tener la *mejor* aproximación posible.

Estos modelos se conocen como redes ya que tiene estructura composicional es decir, son típicamente el resultado de composiciones sucesivas de varios tipos de funciones $f^{(1)}$, $f^{(2)}$, $f^{(3)}, \ldots$.

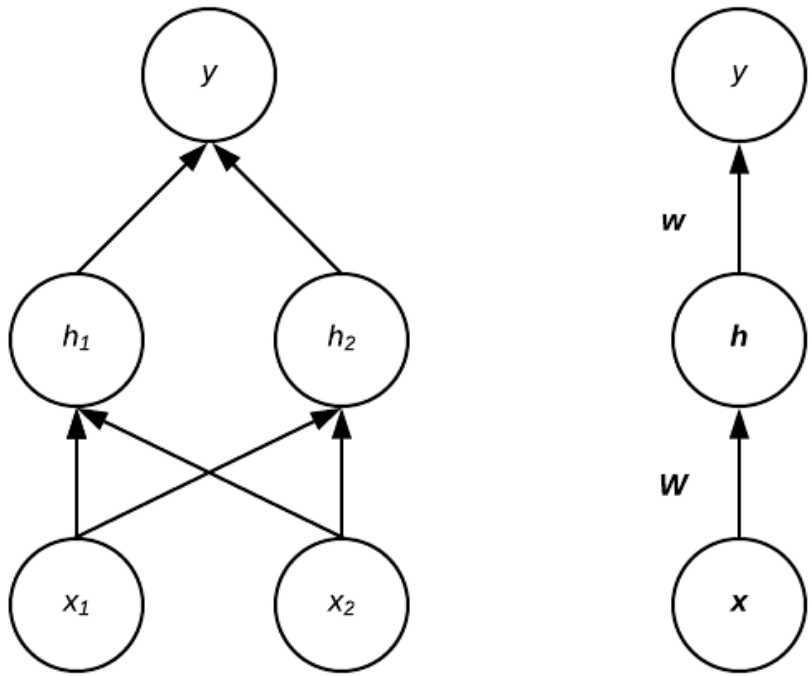

Fig.. Ejemplo de una red neuronal de 1 capa: (*izquierda*). Representación vectorial de las capas. (*derecha*)

Redes Neuronales - El perceptrón

El *perceptrón* corresponde a la forma más básica de una red neuronal, esta recibe un input numérico $x = (x_i)_{i=1}^n \in \mathbb{R}^n$ y computa la suma ponderada $u = x_1w_1 + x_2w_2 + \cdots + w_nx_n + b$ donde $W = (w_i)_{i=1}^n \in \mathbb{R}^n$ corresponden a los **pesos** (weights) y $b \in \mathbb{R}$ el **sesgo** (bias).
Luego, se aplica una **función de activación** f y se entrega una salida $h = f(u)$.

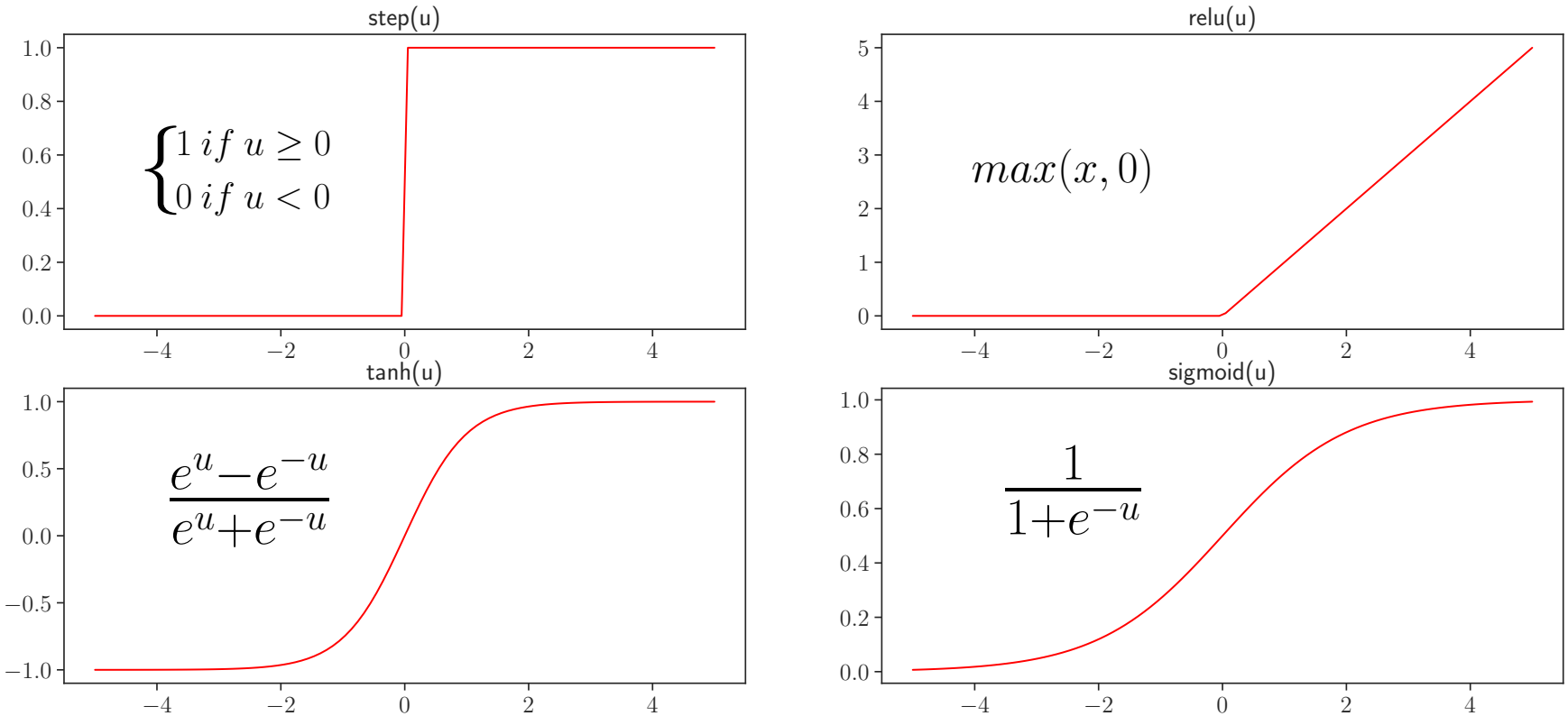

Fig.. Algunos ejemplos de funciones de activación

Redes Neuronales - El perceptrón

Veamos el siguiente ejemplo:

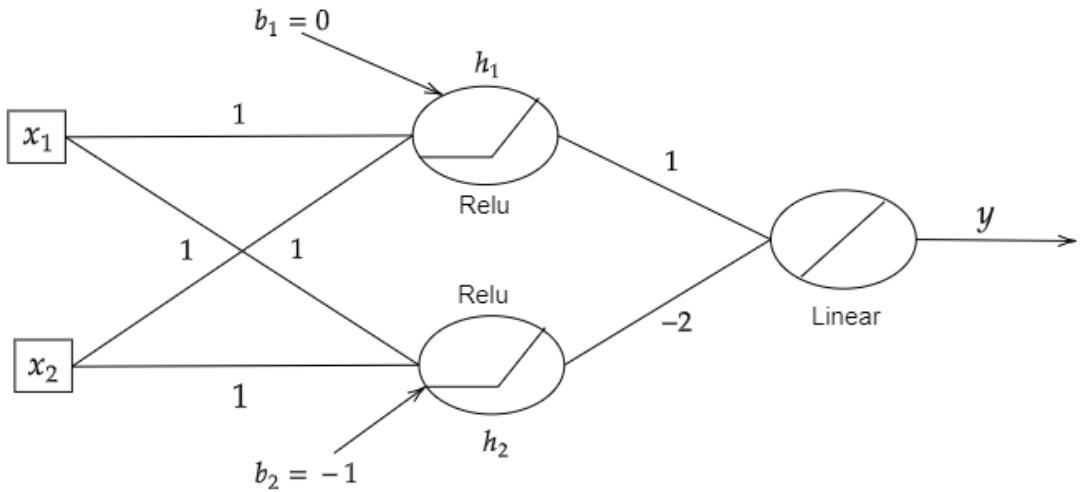

Fig.. Red Neuronal de 2 perceptrones para XOR

XOR operator

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

En su forma matricial

$$(h_1, h_2) = \text{Relu}\left(\begin{pmatrix} x_1 & x_2 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} 0 & -1 \end{pmatrix}\right) \qquad h = \text{Relu}(xW + b)$$

$$y = \begin{pmatrix} h_1 & h_2 \end{pmatrix}\begin{pmatrix} 1 \\ -2 \end{pmatrix} + (0) \qquad y = hU + c$$

Donde U y c corresponden al peso y bias de la última capa respectivamente.

¿Cómo construir el resto de operadores lógicos?

Teorema de Aproximación Universal[1]

Un conjunto de perceptrones conectados unos con otros en secuencia forman arquitecturas capaces de replicar funciones más complejas.
En el contexto de una red de múltiples perceptrones con una sola capa escondida, se tiene el siguiente resultado

Theorem (UAT - Ancho arbitrario)

Sea f^* una función objetivo Borel Medible (por ejemplo $f^* : [0,1]^k \to [0,1]$ continua con $k \in \mathbb{N}$) y sea f una función de activación no polinomial acotada, entonces para todo $\epsilon > 0$ existen W, b, U definidas como antes tales que

$$F(x) = f(xW + b)U, \quad |F(x) - f^*(x)| < \epsilon \quad \forall x \in Dom(f^*)$$

En palabras simples, con una cantidad suficiente de perceptrones (i.e., con una red lo suficientemente *ancha*), es posible aproximar una función objetivo razonable tan finamente como se desee.

[1] (Horniket al., 1989; Cybenko, 1989)

Redes Neuronales - Arquitectura

Si bien el UAT establece que es posible aproximar funciones con precisión arbitraria, tenemos dos problemas:

- no es claro que los pesos se pueden *aprender*,
- no tenemos una cota para la cantidad de neuronas necesarias.

Es por esto que surge la necesidad de arquitecturas más complejas, es decir, con mayor cantidad de capas (mayor profundidad) y no solo neuronas (mayor ancho).

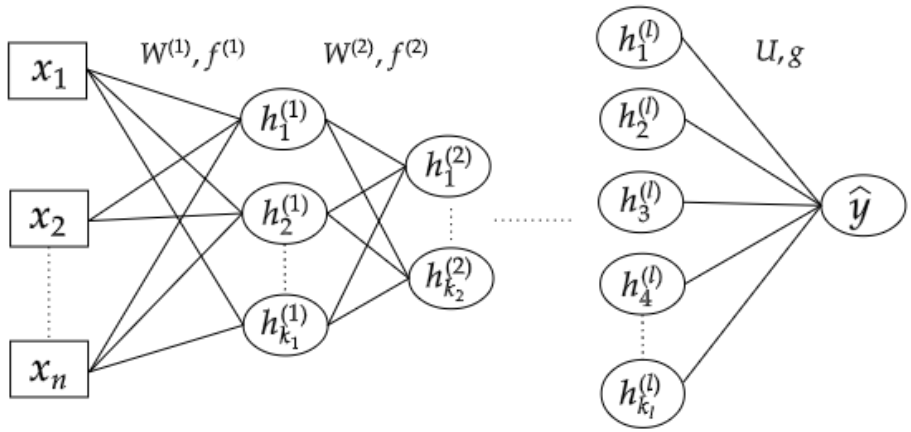

Fig.. Una red con mayor profundidad

Utilizaremos la siguiente notación

$$h^{(k)} = f^{(k)}(h^{(k-1)}W^{(k)} + b^{(k)}) \quad \forall k \in \{1, \ldots, l\}, \quad h^{(0)} = x \tag{1.1}$$

$$\hat{y} = g(h^{(l)}U + c) \tag{1.2}$$

Donde g es la función de la capa de salida y define la **unidad de salida**

Función de costo

Principales diferencias entre los modelos lineales antes vistos y una red neuronal:

- el uso de ciertas funciones de activación hacen que la función de costos **no sea convexa**
- SGD no garantiza optimalidad global
- SGD no garantiza ni siquiera buena solución

Interpretando la red neuronal como un *modelo generativo*, y denotando su ley mediante $p(y|\boldsymbol{x};\boldsymbol{\theta})$, podemos considerar el costo de entrenamiento de la red como la log-verosimilitud negativa del modelo inducido por la red, y estimar sus parámetros mediante máxima verosimilitud.
La **función de costo** que utilizaremos entonces será:

$$J(\boldsymbol{\theta}) = -\log p(y_1, \ldots, y_n|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n; \boldsymbol{\theta})$$

Unidad de salida

Considerando el procesamiento de la red hasta antes de la capa de salida, es posible interpretar que la red está produciendo *características* que luego son combinadas en la última capa para formar una salida. En este sentido, la elección de la **unidad de salida** definirá, en gran parte, la forma que toma la función de costo.
Algunos ejemplos

1. Salida lineal $\hat{\boldsymbol{y}} = h^{(l)}U + c$
   Útil en problemas de regresión. En particular, $\hat{\boldsymbol{y}}$ puede ser la media del modelo propuesto con distrbución Gaussiana condicional $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}|\hat{\boldsymbol{y}}; \boldsymbol{I})$. En dicho caso, la log-verosimilitud será equivalente minimizar el error cuadrático medio.
2. Unidad de salida sigmoidal $\hat{y} = \text{sig}(h^{(l)}U + c)$
   Se utiliza para escenarios de clasificación binaria, cuyo output es $P(y = 1|\boldsymbol{x})$ o la probabilidad de pertenecer a la clase 1.
3. Unidad de output softmax $\hat{y} = \text{softmax}(h^{(l)}U + c)$
   Es una generalización de la función sigmoidal definida como

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}},$$

   y es útil para el caso de clasificación multiclase.