FAMAF Facultad de Matemática, Astronomía, Física y Computación

INTELIGENCIA ARTIFICIAL

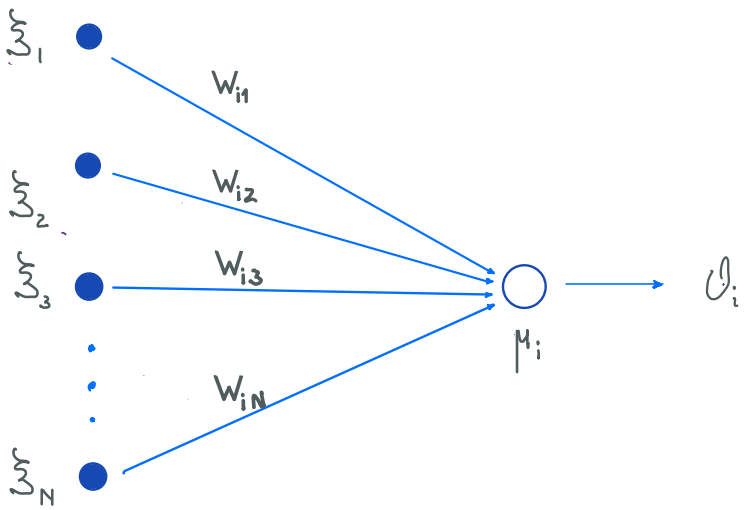
Francisco Tamarit

Clase 2

Facultad de Matemática, Astronomía, Física y Computación
Universidad Nacional de Córdoba

Instituto de Física Enrique Gaviola (UNC y CONICET)

El perceptrón simple



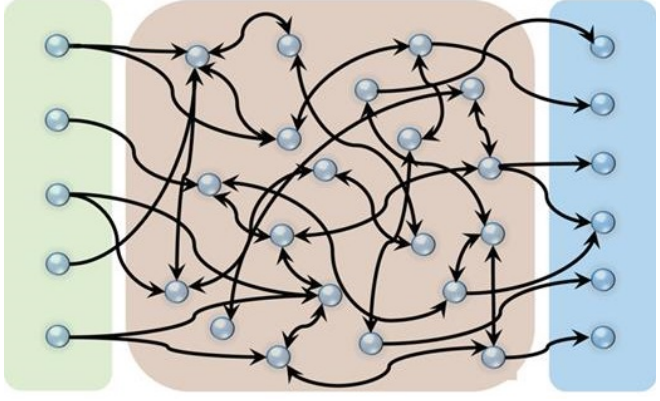
sinapsis — entradas o features — umbral

$$O_i = g\left(\sum_{k=1}^{N} w_{ik}\xi_k - \mu_i\right)$$

$$= g\left(\bar{w}_i \cdot \bar{\xi} - \mu_i\right)$$

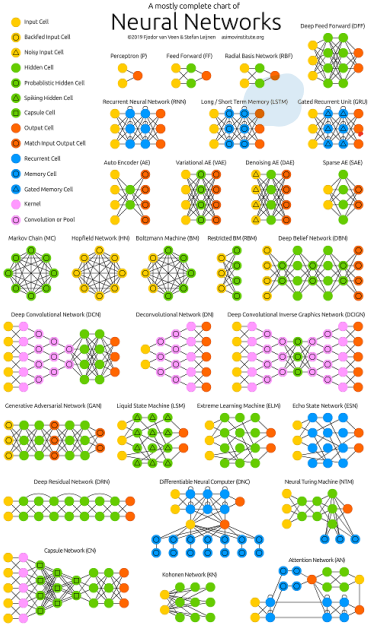
sinapsis — entradas o features — umbral

Las redes neuronales artificiales

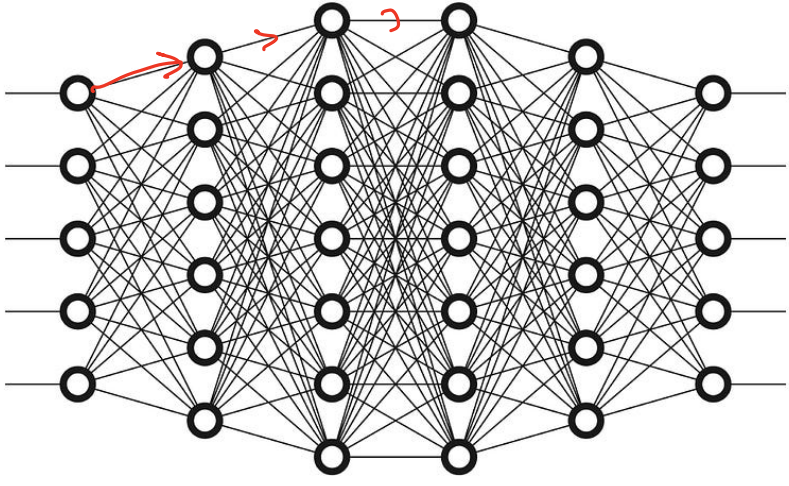


entrada salida

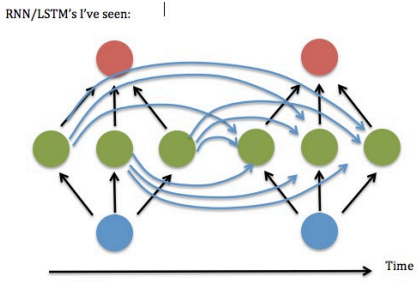
El zoológico de las redes neuronales artificiales

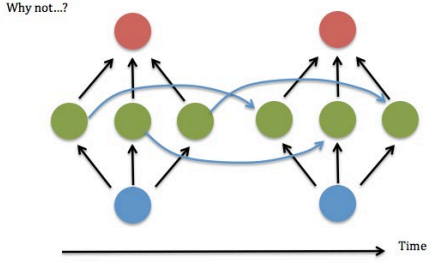


Las redes neuronales artificiales: redes feed-forward

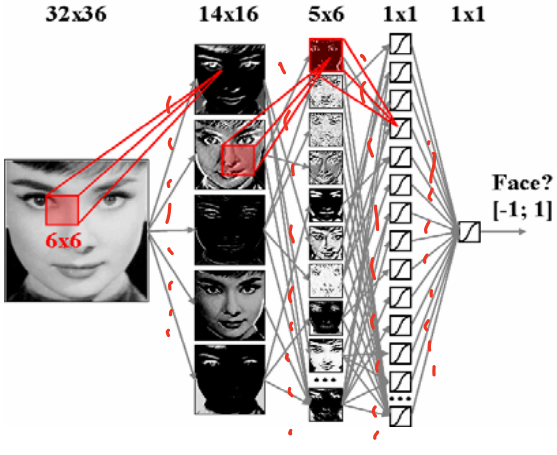


Las redes neuronales artificiales: redes recurrentes

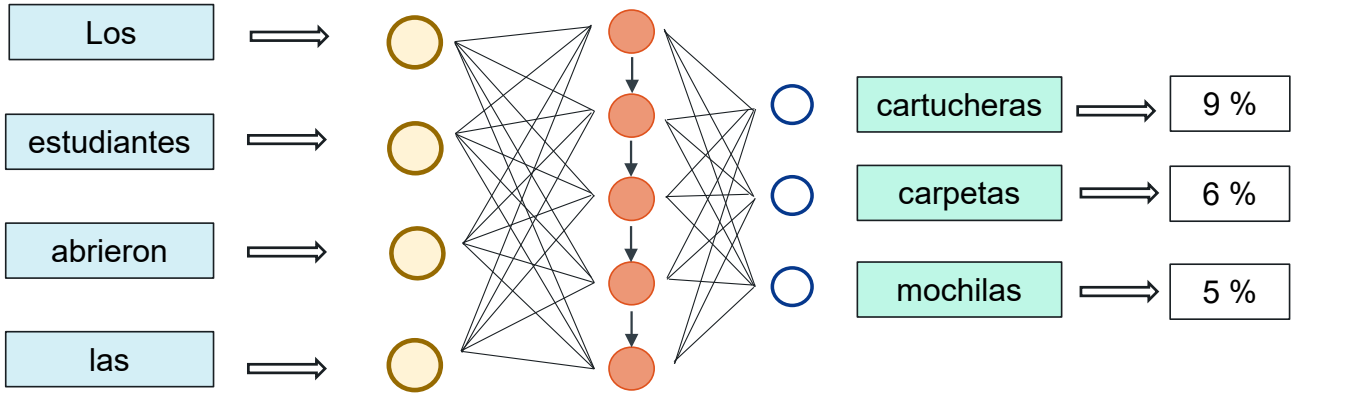




Las redes neuronales artificiales: redes convolucionales



Las redes neuronales recurrentes para lenguaje natural

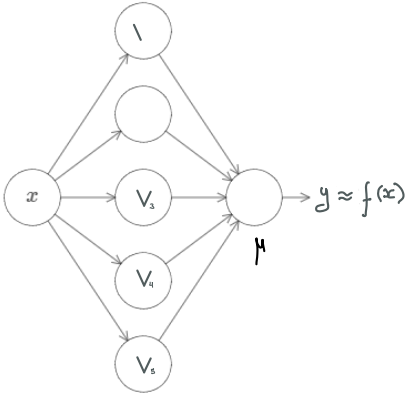


Suponemos que existe cierta función matemática entre Y e X que llamamos f

$$Y = f(X)$$

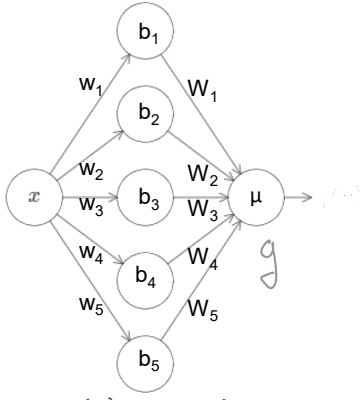
pero que no podemos conocer la función f. Si la conociéramos no necesitaríamos hacer inteligencia artificial. La llamamos *función objetivo*.

Los algoritmos de IA neuronal nos permiten aproximar tan bien como deseamos cualquier función desconocida con una red neuronal, tan compleja como necesitamos. En este caso asumiremos que necesitamos una red neuronal bastante simple que tiene una capa de entrada (*x*), una de salida (*y*) y una capa intermedia u oculta con cinco neuronas, todas simples como las que vimos.



$$\text{ع} = g\left(\left(W_1V_1 + W_2V_2 + W_3V_3 + W_4V_4 + W_5V_5 - \mu\right)\right)$$

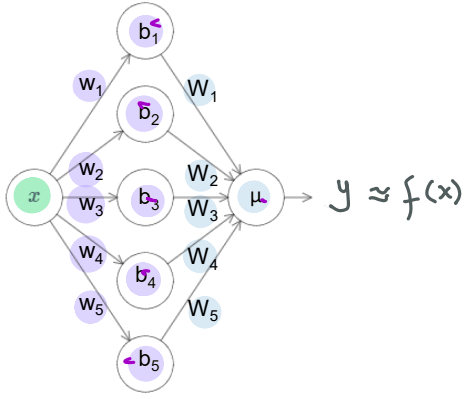
$$= g\left(\sum_j W_j V_j - \mu\right)$$



{w} {b} {W} μ → parámetros

$$y = g\left(\left(W_1 g(\omega_1 x - b_1) + W_2 g(\omega_2 x - b_2) + W_3 g(\omega_3 x - b_3) + W_4 g(\omega_4 x - b_4) + W_5 g(\omega_5 x - b_5) - \mu\right)\right)$$

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



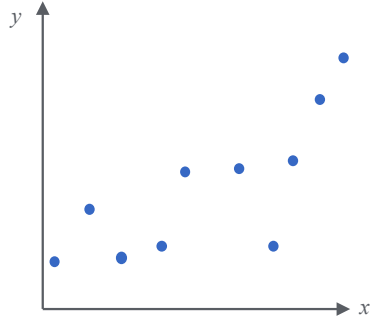
En 1986, Geoffrey Hinton, junto a David Rumenhart y Ronald Williams, introdujeron el algoritmo de *retro-propagación del error* (back-propagation) que nos permite asignar valores adecuados a cada uno de los parámetros a partir del algoritmo de descenso por el gradiente (en nuestro caso tenemos solo *16* parámetro (10 sinapsis y 6 umbrales) pero para CHAT GPT4 tiene aproximadamente 100.000.000.000.000 parámetros).

El método que nos permite encontrar el o los parámetros que minimizan el ECM se llama el método de los mínimos cuadrados y fue introducido por Karl Friedrich Gauss en 1794 pero fue publicado recién en 1809.

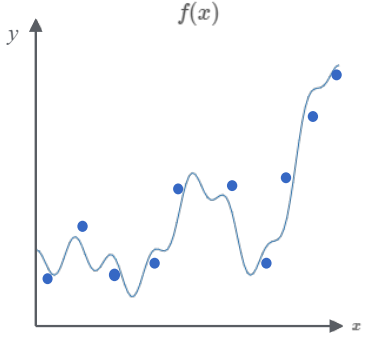
Se trata de un método de optimización que nos permite, en forma algorítmica, probar diferentes valores de *g* hasta que llegamos al valor óptimo. Esto tiene un método definido.

Este ejemplo que acabamos de ver, sobre el experimento de medir el tiempo que le lleva a la pelota alcanzar el suelo, es un *método transparente*, pues si alguien pregunta, podemos contarle que la relación entre *Y* e *X* es una raíz cuadrada. Pero no siempre tenemos ese conocimiento a priori que nos da la expresión matemática entre ambas variables.

Supongamos ahora que estudiando un fenómeno de alguna disciplina más complicada que la física (por ejemplo, podría ser la climatología o la medicina) tenemos una variable dependiente *y* y una variable independiente *x* pero no podemos conocer la relación funcional. Sin embargo midiendo observamos que hay una relación.



Supongamos ahora que estudiando un fenómeno de alguna disciplina más complicada que la física (por ejemplo, podría ser la climatología o la medicina) tenemos una variable dependiente *y* y una variable independiente *x* pero no podemos conocer la relación funcional. Sin embargo midiendo observamos que hay una relación.



Ya tenemos entonces la forma de la función que aproximará (es una regresión fuertemente no lineal) a nuestra función objetivo desconocida f(x). Podemos ahora definir el error cuadrático medio:

$$\text{ECM} = \frac{1}{10}((y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + \ \ldots \ + (y_{10} - \hat{y}_{10})^2)$$

$$= \frac{1}{10}\sum_{i=1}^{10} \ (y_i - \hat{y}_i)^2$$

EL MÉTODO DE BACK-PROPAGATION

1. Inicialmente le asignamos a cada uno de los *16* parámetros un valor aleatorio, por ejemplo, entre *-1* y *+1*, con probabilidad uniforme.
2. Le mostramos a la red el primer ejemplo sacado del conjunto de entrenamiento y calculamos el valor de salida y^1.
3. Con la salida y^1 y el resultado correcto obtenido del conjunto de entrenamiento calculamos el error cuadrático medio ECM, el cual tiene dentro de sí, todos los parámetros (sinapsis y umbrales) que definen nuestra red.

4. Le aplicamos la técnica del descenso por el gradiente y corregimos levemente uno a uno todos los parámetros desde atrás hacia delante de manera que al final la red de una respuesta más cerca a la correcta cuando le volvamos a mostrar el primer ejemplo del conjunto de entrenamiento.

$$W_i^{\,nuevo} = W_i^{\,anterior} + \Delta W_i \qquad \Delta W_i = -\eta_i \frac{\partial ECM}{\partial Wi}$$

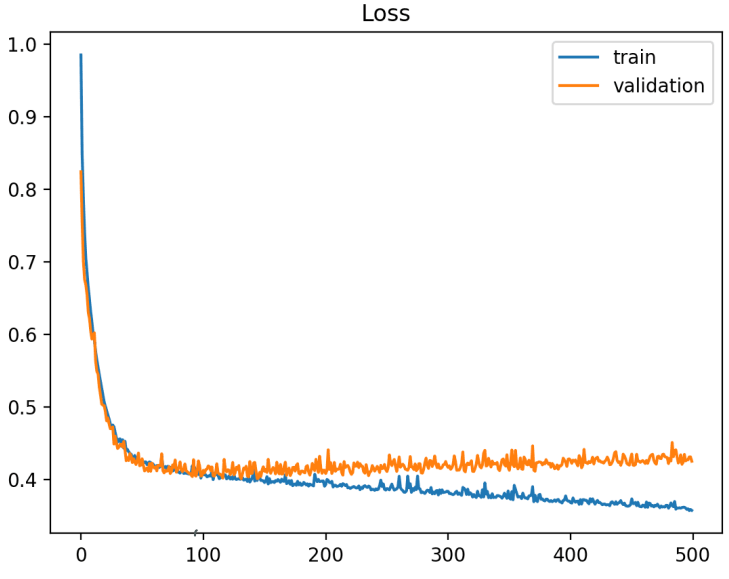
$$b_i^{\,nuevo} = b_i^{\,anterior} + \Delta b_i \qquad \Delta b_i = -\eta_i \frac{\partial ECM}{\partial bi}$$

- η se denomina razón de aprendizaje y modera los cambios que sufren los parámetros
- η toma valores pequeños, típicamente del orden de 0,01, y es el mismo para todas sinapsis y todos los umbrales.

5. Repetimos este preciso para cada uno de los ejemplos.
6. Cuando terminamos con todos los ejemplo (llamamos a esto una época) volvemos a repetir y repetir hasta que alcanzamos un valor del ECM aceptable.

Dividimos el conjunto de entrenamiento en dos partes, una para aprender y otra para validar si anda bien. En cada época evaluamos el funcionamiento de la red, o sea, su ECM, sobre el conjunto de entrenamiento y el de validación.

Si hemos hecho las cosas bien, la red que aprendió el conjunto de entrenamiento también sabrá resolver adecuadamente los valores de validación y por tanto, los valores que nunca medimos.


Loss
train
validation
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
100
200
300
400
500