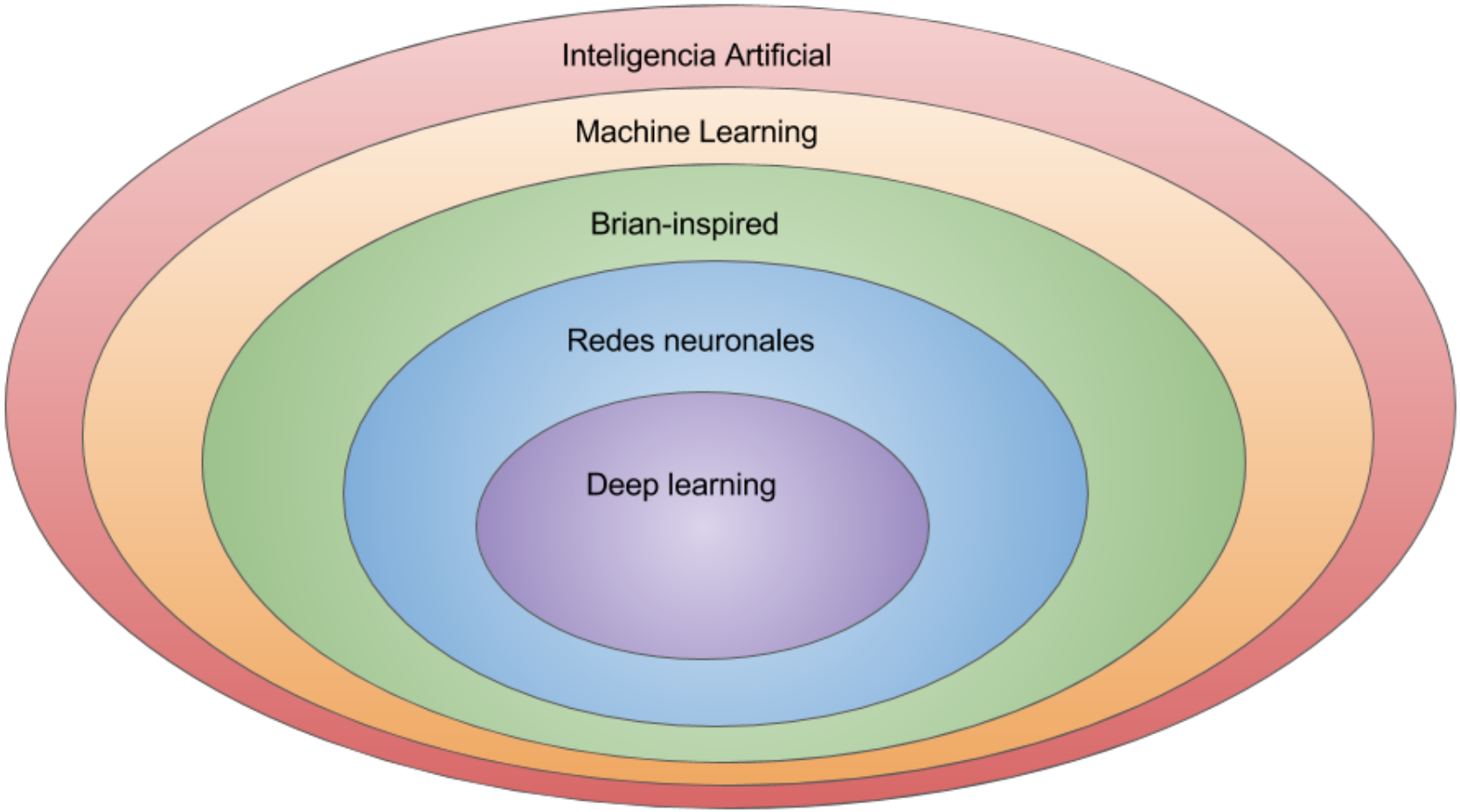
Redes Neuronales

Introducción

Rosana Matuk

Segundo Cuatrimestre 2018

Deep learning en el contexto de la inteligencia artificial



Contexto

Inteligencia artificial (AI)

La ciencia e ingeniería de crear máquinas inteligentes que tengan la habilidad de conseguir objetivos como hacen los humanos (John McCarthy, 1956).

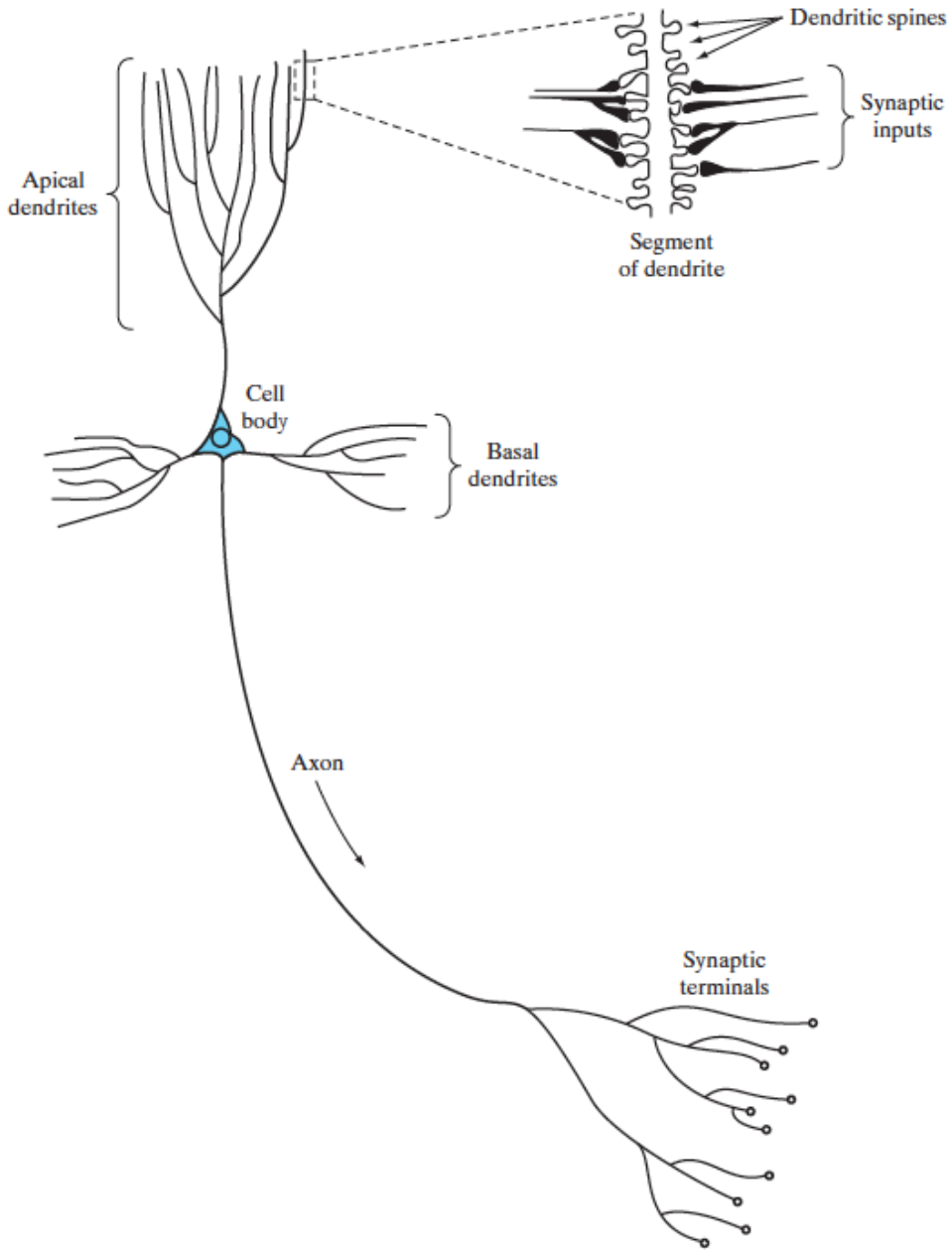
Machine learning

El algoritmo aprende de un conjunto de datos, via un proceso llamado entrenamiento, a resolver nuevos problemas (Arthur Samuel,1959).

Brian-inspired

Estudia el diseño de algoritmos que reflejen en algún aspecto de su forma básica o funcionalidad, el funcionamiento del cerebro.

Neurona biológica



Neurona biológica

Las unidades de cómputo del cerebro son células llamadas neuronas.
Una neurona, recibe señales a través de sus dendritas, procesa las señales recibidas en el cuerpo celular, y emite una señal por el axón, si la excitación total recibida supera cierto umbral.

Contexto (cont.)

Redes neuronales

Toman su inspiración de la noción que un cómputo en una neurona involucra una sumatoria pesada de los valores de entrada, a través de las dendritas, y la combinación y procesamiento de esos valores en el cuerpo de la neurona. La operación en el cuerpo de la neurona, parece ser una función no lineal de los inputs, y hace que una neurona genere una salida, si la combinacion de las entradas superan cierto umbral (Frank Rosenblatt, 1957).

Deep learning

Las redes neuronales originales tenían muy pocas capas en la red. En las redes neuronales profundas hay muchas más capas (Geoffrey Hinton, 2005).

Paradigmas de aprendizaje

Aprendizaje supervisado

Dado un conjunto de datos de entrenamiento consistente de pares entrada-salida, encontrar una función que dada una nueva entrada prediga la salida esperada.

Aprendizaje no supervisado

Dado un conjunto de datos de entrenamiento consistentes de inputs no etiquetados (no se tiene una salida esperada), inferir una función que describa la estructura oculta de los datos (por ej, clusters).

Aprendizaje semi-supervisado

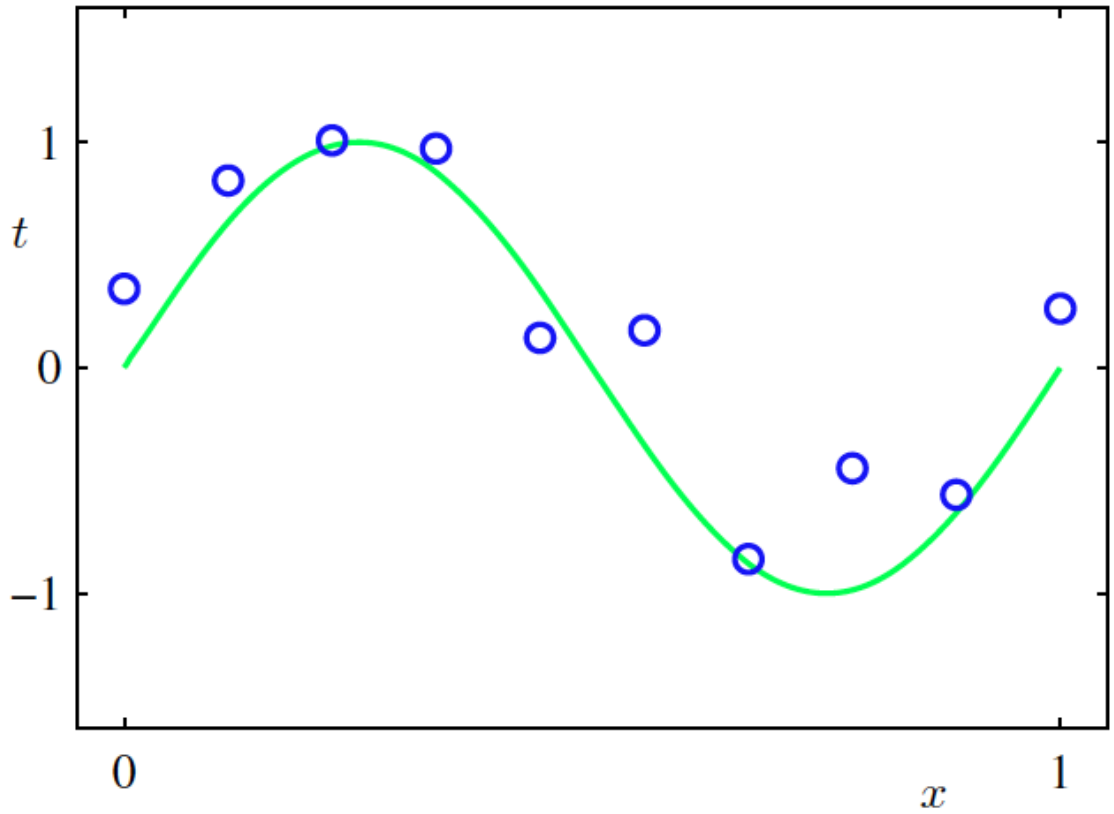
Combinación de las dos anteriores. Solo un pequeño subconjunto de los datos de entrenamiento está etiquetado (por ej, usamos los datos no etiquetados para detectar clusteres, y usamos el conjunto de datos etiquetados para etiquetar los clusteres).

Paradigmas de aprendizaje (cont.)

Aprendizaje por refuerzo

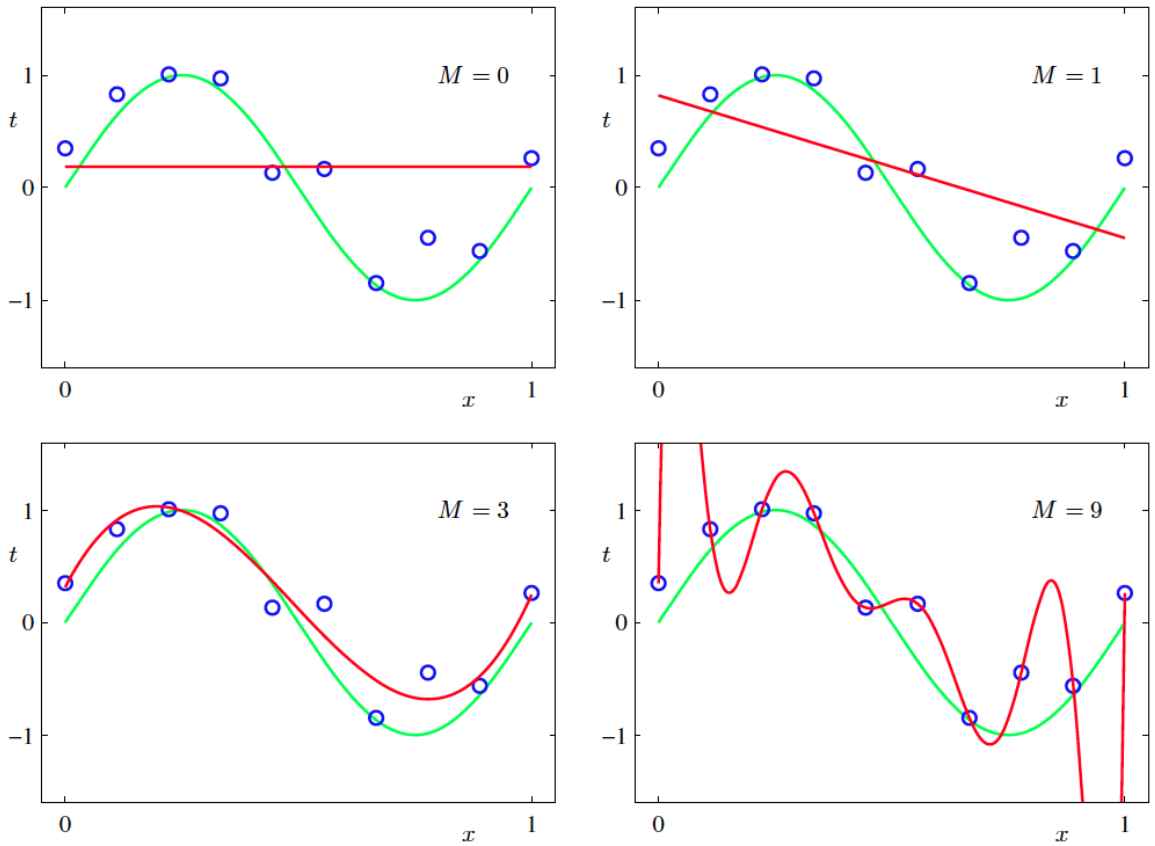
Dado un input, decide una acción a realizar, y recibe una recompensa (positiva o negativa). El objetivo es entrenar la red para que tome decisiones que maximicen la función de premio. El proceso de entrenamiento debe balancear la exploración (intentar nuevas acciones) y la explotación (usar acciones conocidas que tengan premios altos). Ejemplo: programas de inteligencia artificial que ganan juegos (ajedrez, go, etc).

Objetivo en machine learning: Aprender de los datos



La curva verde muestra la función $\sin(2\pi x)$. Los puntos azules son muestras de la función seno, que tienen ruido, y nos proveen de dicho dataset, pero no nos dicen la función que los origina. Nuestro objetivo es, a partir del dataset (puntos azules) aproximar la función desconocida.

Underfitting, Generalización, Overfitting



Si ponemos muy pocas neuronas, la red no tendrá suficiente poder de cómputo para poder modelar la función (**underfitting**). Si ponemos demasiadas neuronas, la red tendrá poder para modelar funciones más complejas que la función verde, y debemos tener cuidado de no caer en **overfitting** (una función que pase por las muestras de entrenamiento, pero que no generalice la función).

Función de costo

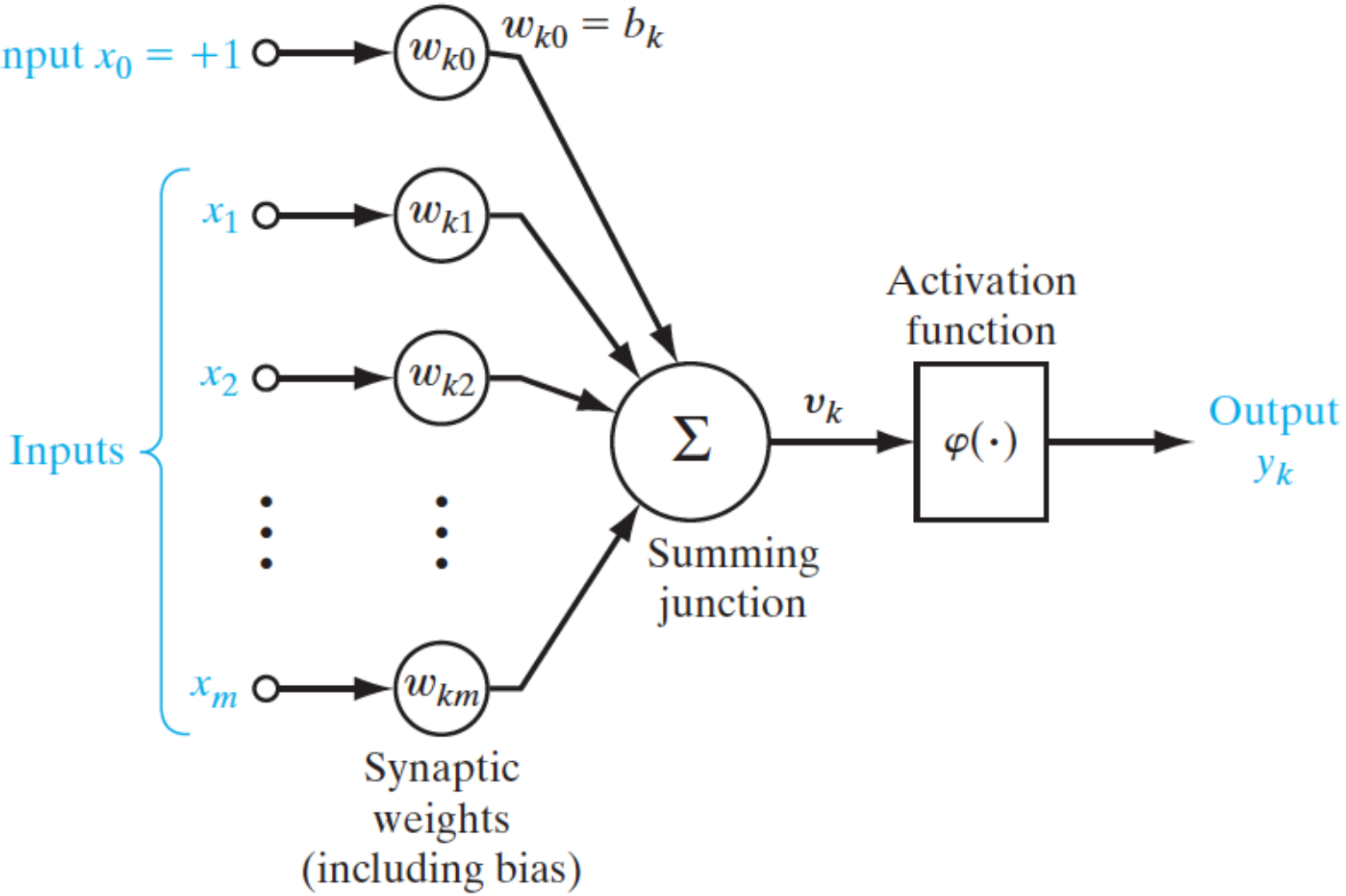
Nuestro objetivo es **generalizar**, y entonces, tenemos que hacer predicciones con el menor número posible de errores. Entonces, necesitamos una métrica del error, llamada función de error o función de costo, y tratamos de encontrarle el mínimo, mediante un método de optimización (por ej, gradiente descendiente).

Por ejemplo, una función de costo comúnmente utilizada es la *suma de los errores cuadráticos*:

$$E = \frac{1}{2} \sum_{i\mu} (\zeta_i^\mu - O_i^\mu)^2$$

donde ζ es el target esperado, O es el output de la red, μ es el patrón de entrada, y i es la unidad de salida.

Red Neuronal Artificial *Perceptrón Simple*



Red neuronal feed-forward de una sola capa.

Funciones de activación

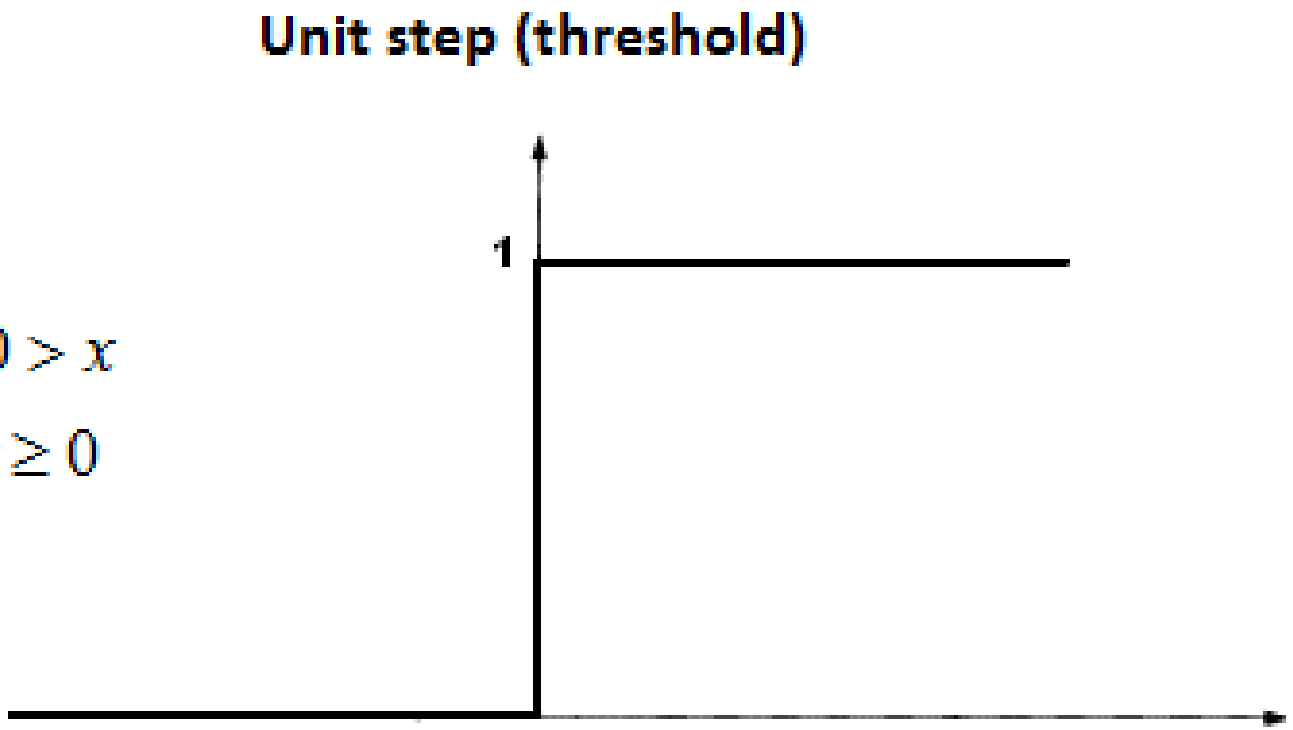
La *función de activación*, emula el procesamiento dentro del cuerpo de la neurona, de las entradas recibidas por sus dendritas.

Algunas funciones de activación clásicamente usadas en redes neuronales:

- *Escalón*
- *Lineal*
- *No lineal:* Sigmoidea

Función escalón

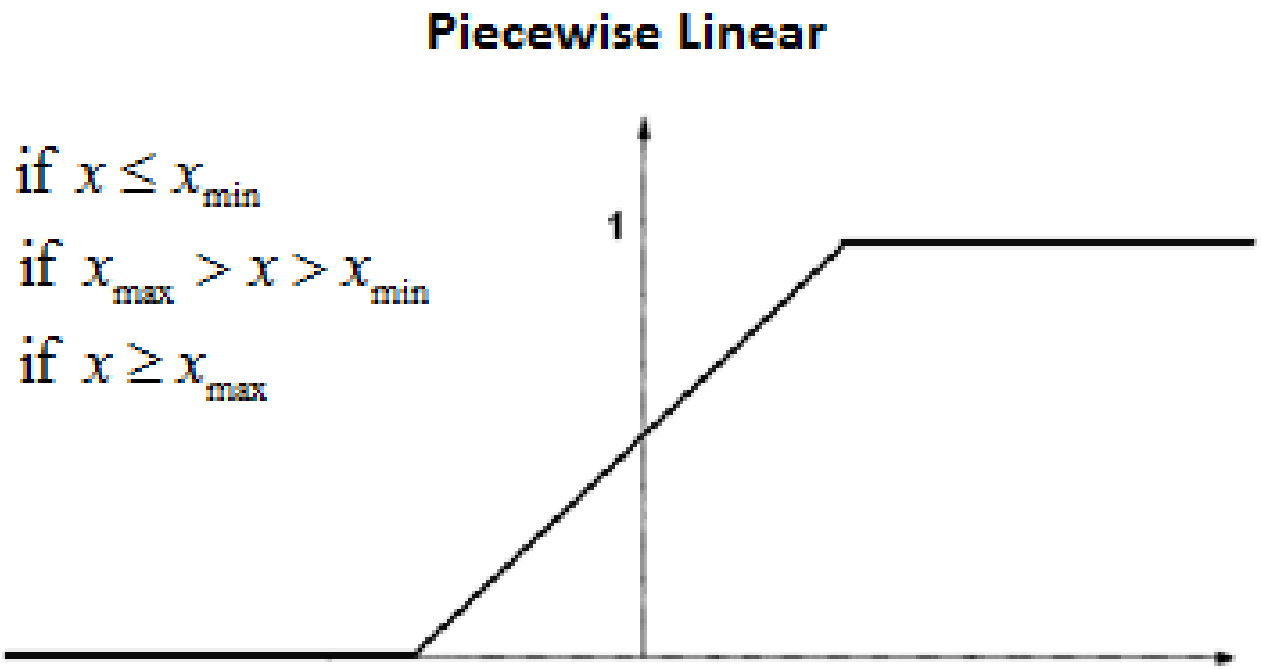
$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



El output es binario, y depende de si el input total es mayor que o menor que cierto valor umbral. Equivaldría sólo a modelar si la neurona dispara o no dispara.

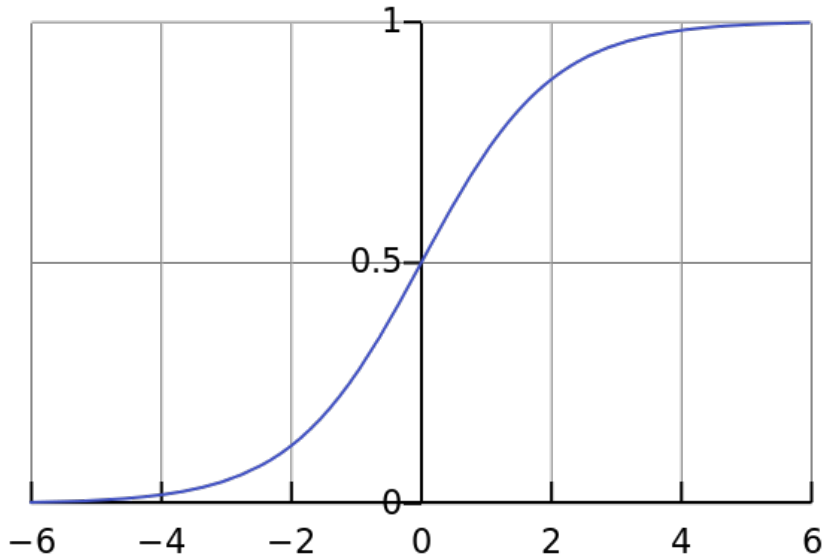
Función lineal de a tramos

$$f(x) = \begin{cases} 0 & \text{if } x \leq x_{\min} \\ mx + b & \text{if } x_{\max} > x > x_{\min} \\ 1 & \text{if } x \geq x_{\max} \end{cases}$$



El output es lineal entre ciertos umbrales superior e inferior. Pasados dichos umbrales, la neurona se satura, y su salida es constante. Equivaldría a modelar la saturación de las neuronas biológicas, y la fuerza de la señal de salida.

Función logística

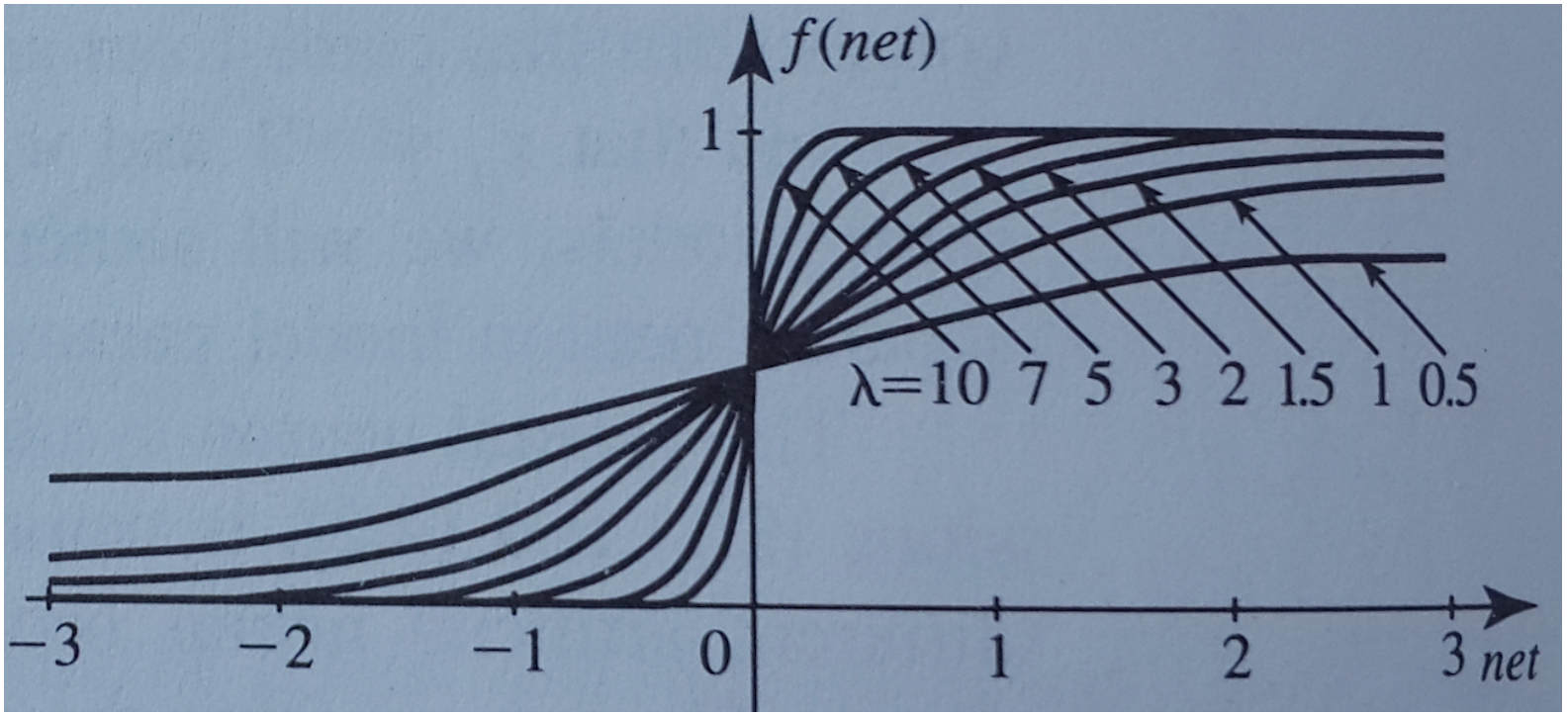


La *función logística* es una función sigmoidea (o sea, cuyo gráfico tiene forma de S), con ecuación:

$$f(x) = \frac{L}{1 + e^{-\lambda(x - x_0)}}$$

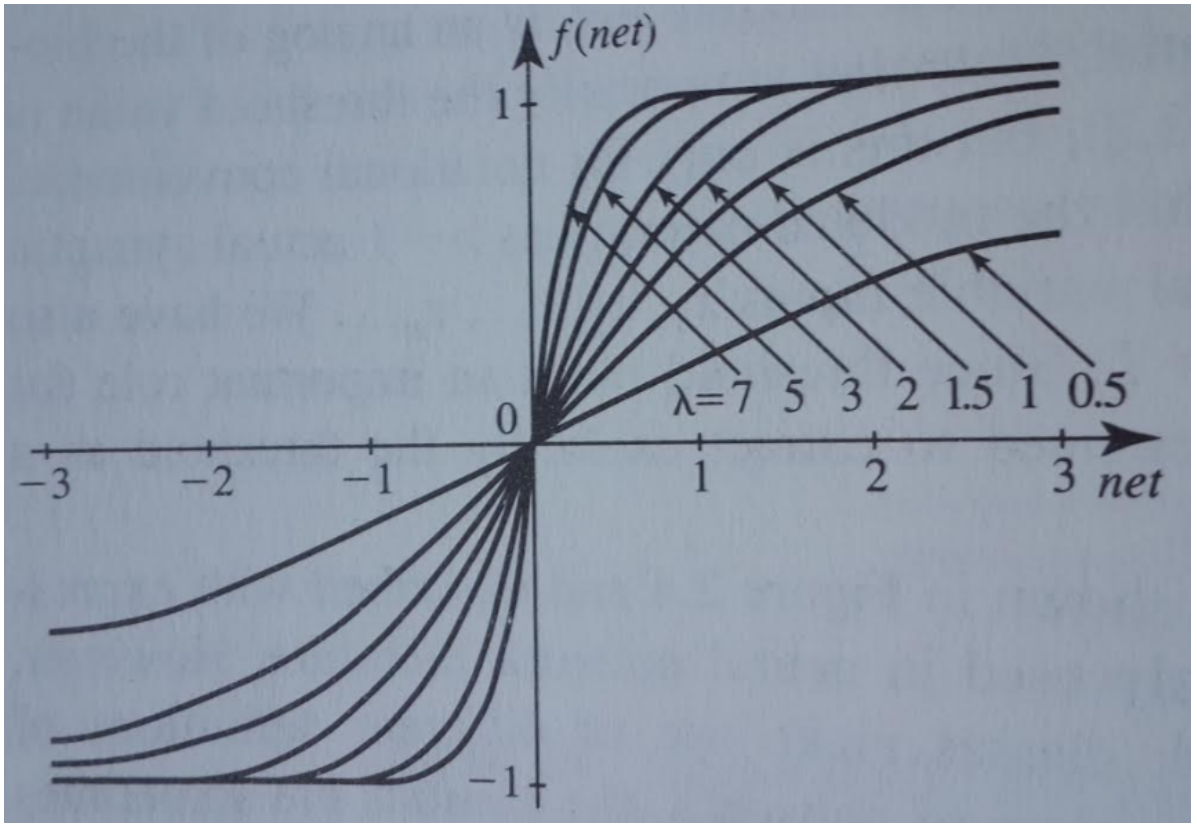
donde x_0 es el valor en el eje x del punto medio de la sigmoidea, L es el valor máximo de curva, y λ es la pendiente de la curva.
La *función logística standard* tiene parámetros $L = 1$, $\lambda = 1$, $x_0 = 0$, y es la de la figura.

Función logística



A medida que aumenta λ, la función logística se va asemejando a la escalón. A medida que disminuye λ, se va asemejando a una función lineal de a tramos. Esto nos permite modelar las funciones anteriores, teniendo además la importante propiedad de ser diferenciable.

Transformaciones de la función logística



La función logística, la podemos escalar y trasladar, y de esa forma, su imagen estaría en cualquier rango deseado. Por ejemplo, si queremos que la imagen esté entre -1 y +1:

$$f(net) = \frac{2}{1+e^{(-\lambda net)}} - 1$$

Función tangente hiperbólica

La función tangente hiperbólica, es otra función sigmoidea ampliamente utilizada, cuya imagen está entre -1 y +1.

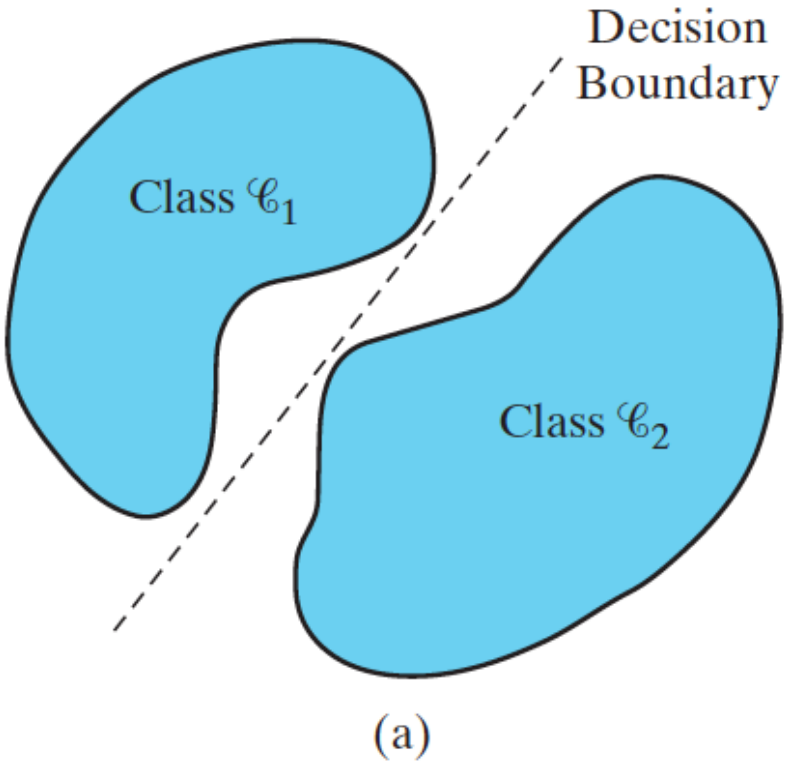
Las funciones tangente hiperbólica y logística son esencialmente equivalentes, ya que una puede ser obtenida de la otra, por translación y escalado:

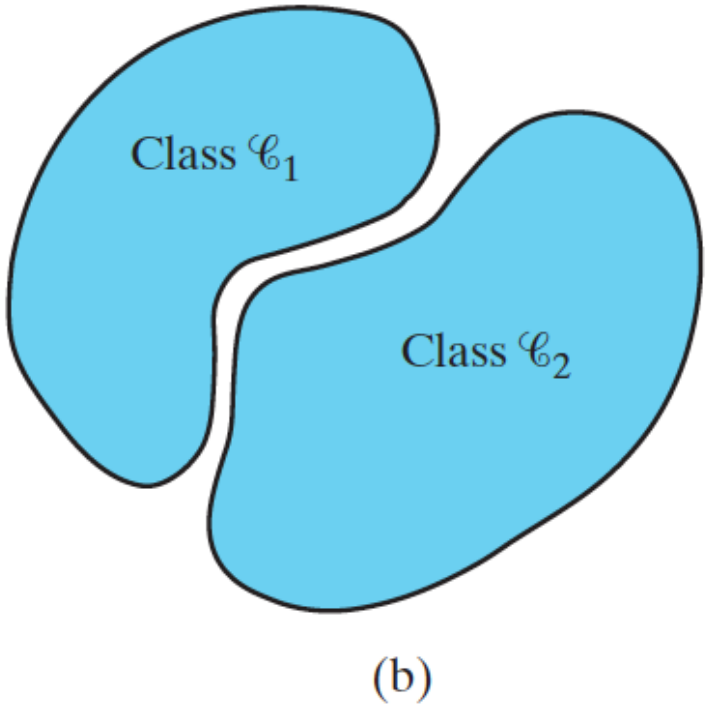
$$\frac{1}{1+e^{(-x)}} - \frac{1}{2} = \frac{1}{2}\tanh(\frac{x}{2})$$

Limitaciones del perceptrón simple

El perceptrón simple sólo puede resolver problemas en los cuales los datos sean linealmente separables. Geométricamente, esta condición describe la situación en la cual existe un hiperplano en grado de separar, en el espacio vectorial de los inputs, las entradas con salidas positivas de las entradas con salidas negativas.

Separabilidad lineal

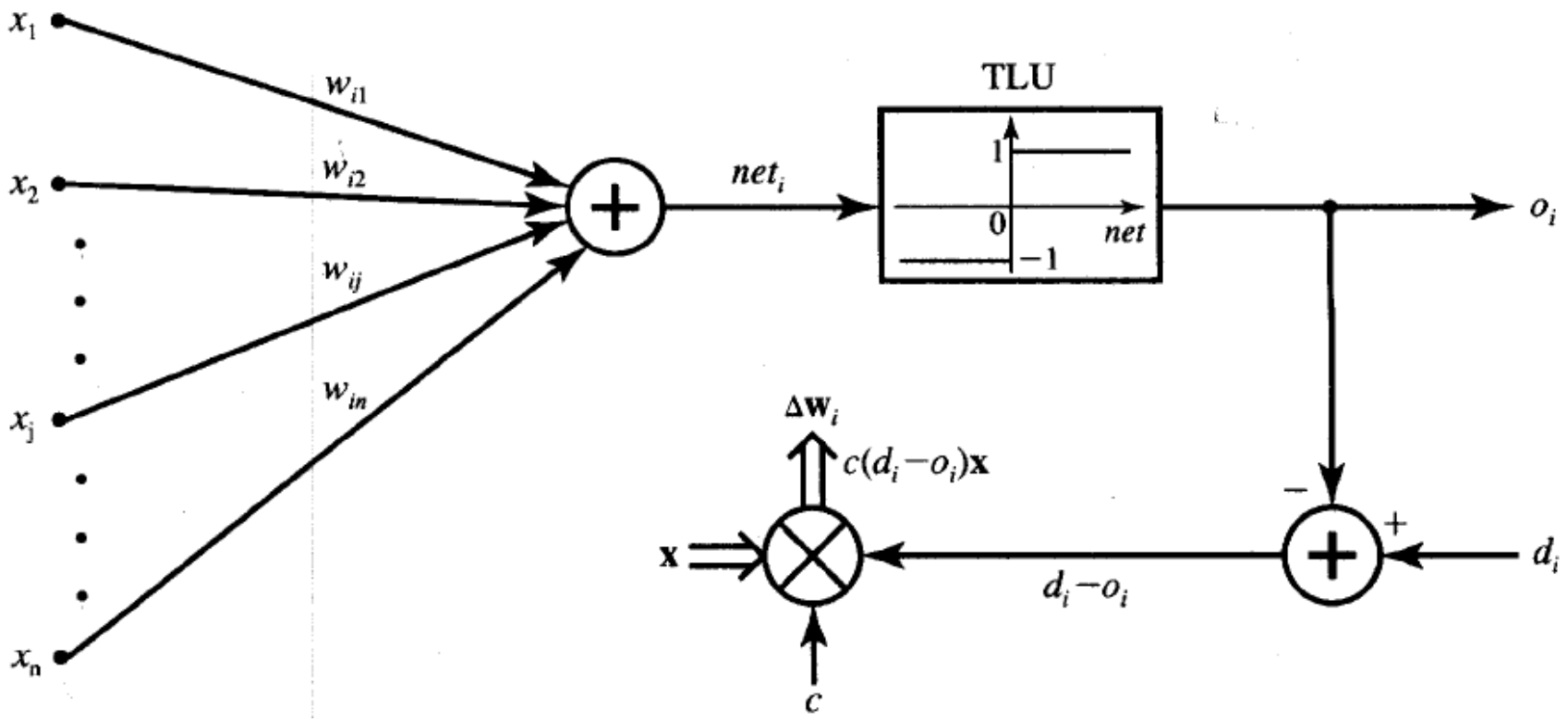




(a) Datos linealmente separables

(b) Datos no linealmente separables

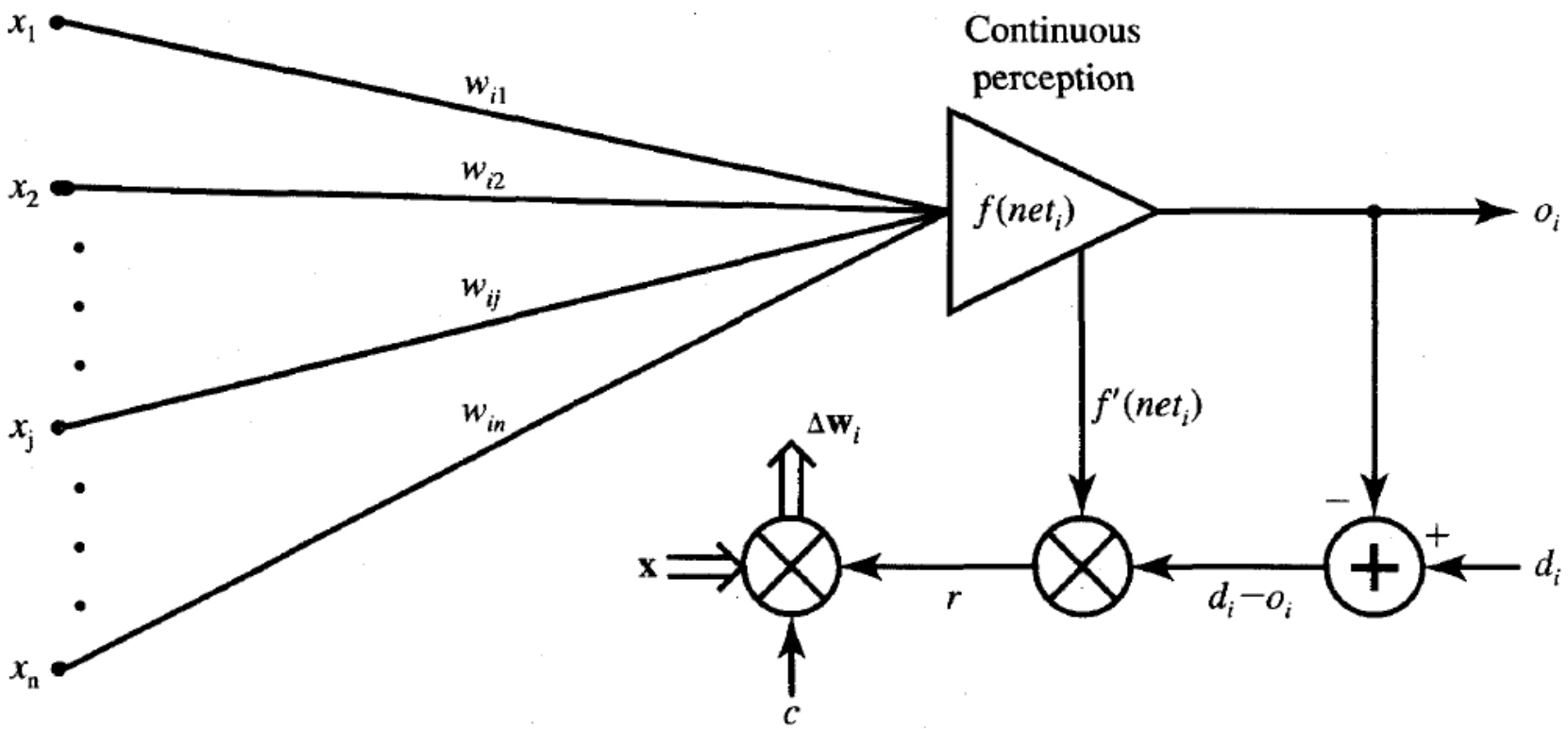
Regla de aprendizaje del Perceptrón Simple Escalón



$$\Delta \mathbf{w}_i = c\left[d_i - \operatorname{sgn}\left(\mathbf{w}_i^t \mathbf{x}\right)\right]\mathbf{x}$$

Notación: w_i es el vector de pesos de la neurona i, c es el factor de aprendizaje, d_i es la salida deseada de la neurona i, o_i es el output real de la neurona i, x es el patrón de entrada, TLU (Threshold Logic Unit) es la función de activación escalón, net_i es $w_i^t x$.

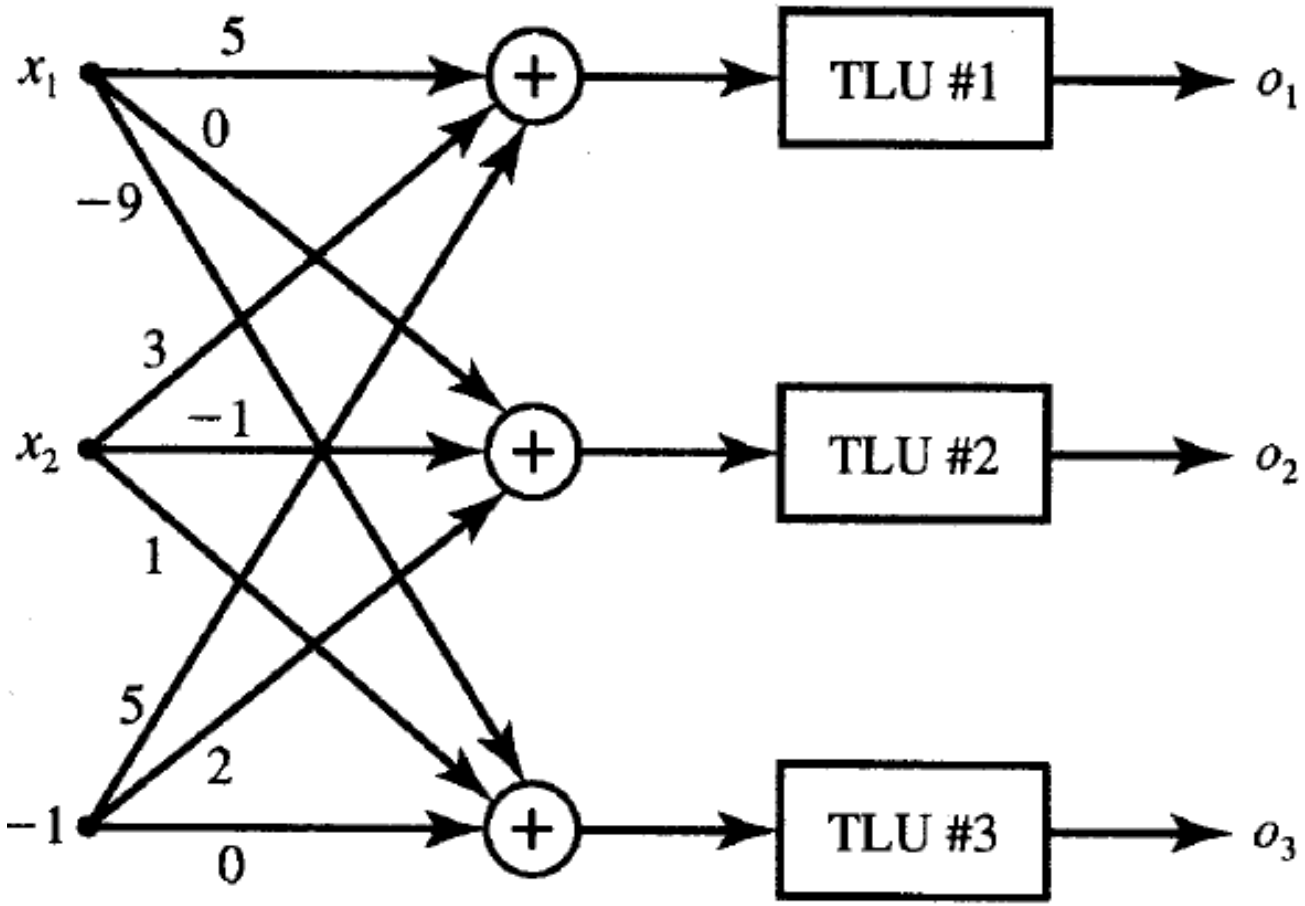
Regla de aprendizaje del Perceptrón Simple Continuo



$$\Delta \mathbf{w}_i = c(d_i - o_i) f'(\mathbf{net}_i)\mathbf{x}$$

Notación: w_i es el vector de pesos de la neurona i, c es el factor de aprendizaje, d_i es la salida deseada de la neurona i, o_i es el output real de la neurona i, x es el patrón de entrada, f es una función activación derivable, net_i es $w_i^t x$.

Ejemplo perceptrón de 3 unidades de salida



Pseudocódigo super-simplificado del perceptrón simple

1) *Inicialización.* Inicializar pesos de la red en forma random.
2) *Entrenamiento.* Mientras no se cumpla condicion de corte, hacer:
2.a) *Activación.* Elegir un patrón del conjunto de entrenamiento.
2.b) *Cómputo de la respuesta actual.* Calcular en forma forward el output de la red neuronal.
2.c) *Adaptación de los pesos.* Calcular el error de la red con respecto a la salida esperada, y actualizar los pesos de la red.

¿De dónde sale la regla Δw del perceptrón simple?

La derivación de la regla Δw de modificación de los pesos de la red en el perceptrón simple, depende de la función de activación utilizada:

- *Función activación escalón:* como la función escalón no es diferenciable, la regla surge por aprendizaje hebbiano.
- *Función de activación diferenciable:* surge de la minimización de la función de costo mediante algún algoritmo de optimización.

Pseudocódigo del perceptrón simple escalón

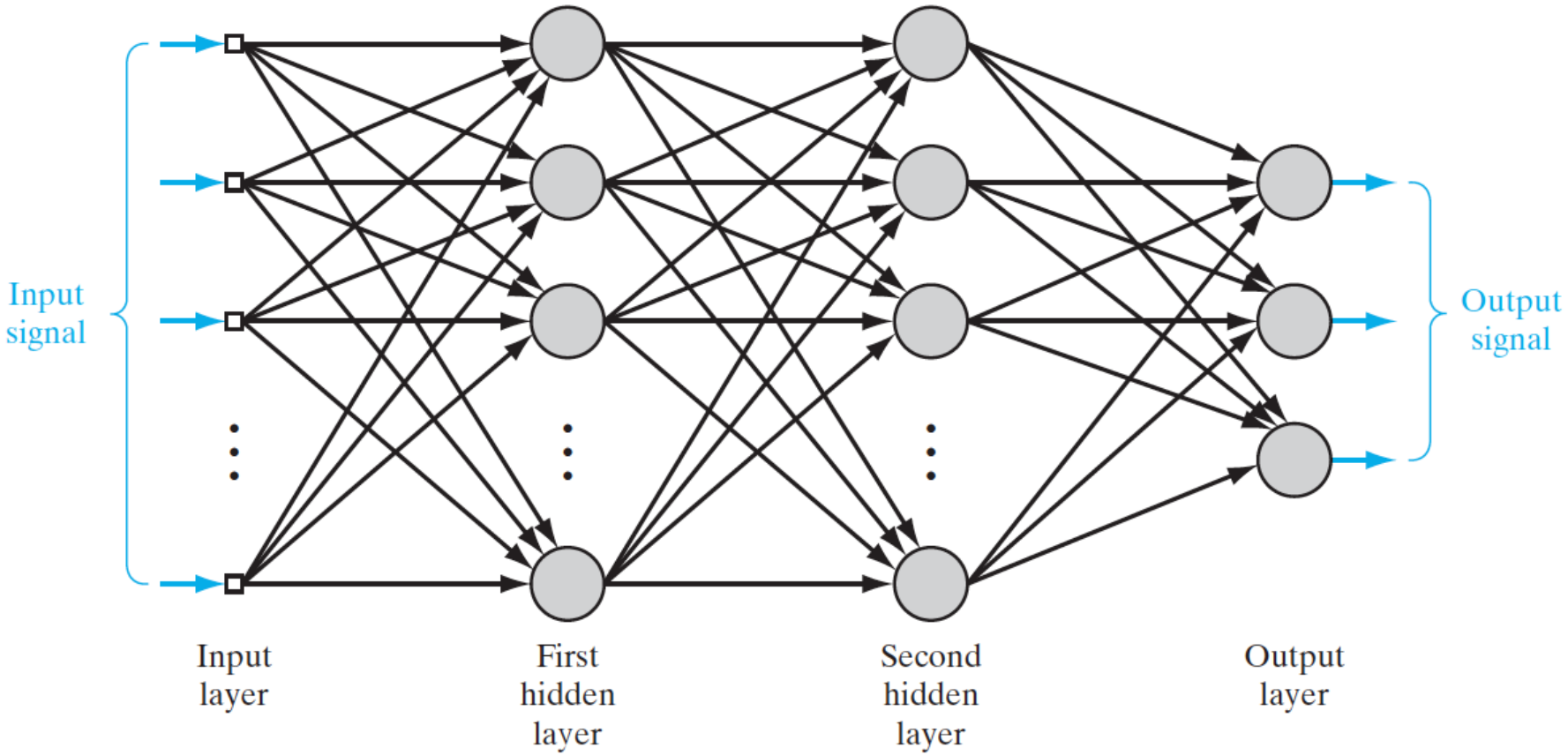
TABLE 1.1 Summary of the Perceptron Convergence Algorithm

Variables and Parameters:

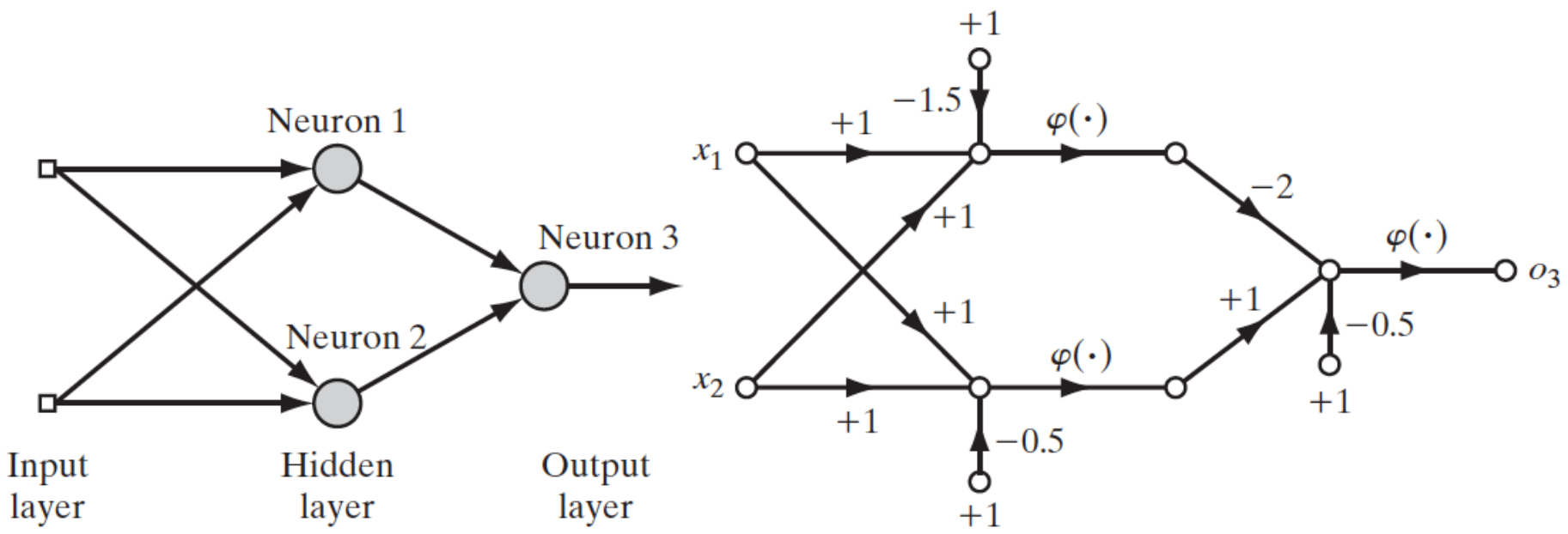
$\mathbf{x}(n)$ = $(m+1)$-by-1 input vector
$= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$
$\mathbf{w}(n)$ = $(m+1)$-by-1 weight vector
$= [b, w_1(n), w_2(n), \ldots, w_m(n)]^T$
b = bias
$y(n)$ = actual response (quantized)
$d(n)$ = desired response
η = learning-rate parameter, a positive constant less than unity

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, \ldots$.
2. *Activation.* At time-step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. *Computation of Actual Response.* Compute the actual response of the perceptron as
$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$
where sgn(·) is the signum function.
4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$
where
$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$
5. *Continuation.* Increment time step n by one and go back to step 2.

Red neuronal feedforward multicapa



Red neuronal feedforward multicapa: XOR



Poder de las redes feedforward multicapa

Las redes feedforward multicapa, con funciones de activación que cumplan ciertas condiciones (ver teorema), son aproximadores universales de funciones medibles de Borel (las funciones continuas en un subconjunto cerrado y acotado de R^n son de Borel). Las funciones de activación sigmoideas cumplen las condiciones del teorema.

Teorema de la Aproximación Universal (Cybenko, 1988; Hornik et al., 1989; Funahashi, 1989)

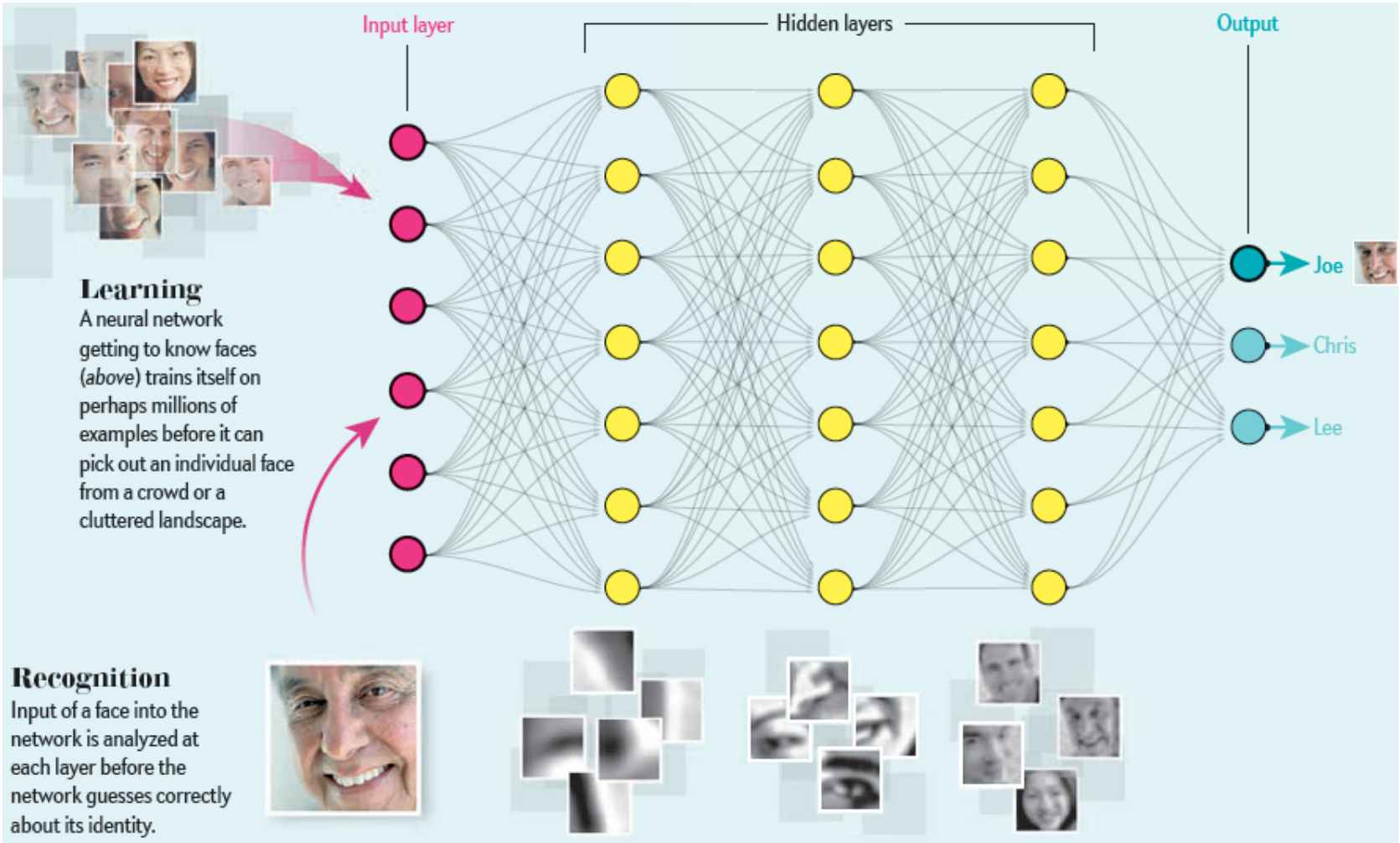
Una red feedforward multicapa, con una capa de salida con función de activación lineal, y al menos una capa oculta con funciones de activación continuas, no constantes, acotadas, y monotonamente crecientes, puede aproximar cualquier función medible de Borel, de un espacio de dimensión finita a otro con cualquier grado de error distinto de cero que se desee, si tiene la suficiente cantidad de neuronas ocultas.

Poder de las redes feedforward multicapa (cont.)

Atención: El Teorema de la Aproximación Universal es un teorema de existencia!

El Teorema de la Aproximación Universal, nos dice sólo que **existe** una red neuronal feed-forward de una sola capa oculta, que puede aproximar cualquier función continua en un subconjunto cerrado y acotado de R^n, pero no nos dice la cantidad de neuronas ocultas que debe tener, ni cómo obtener los pesos de dicha red neuronal.

Poder de representación en redes multicapas

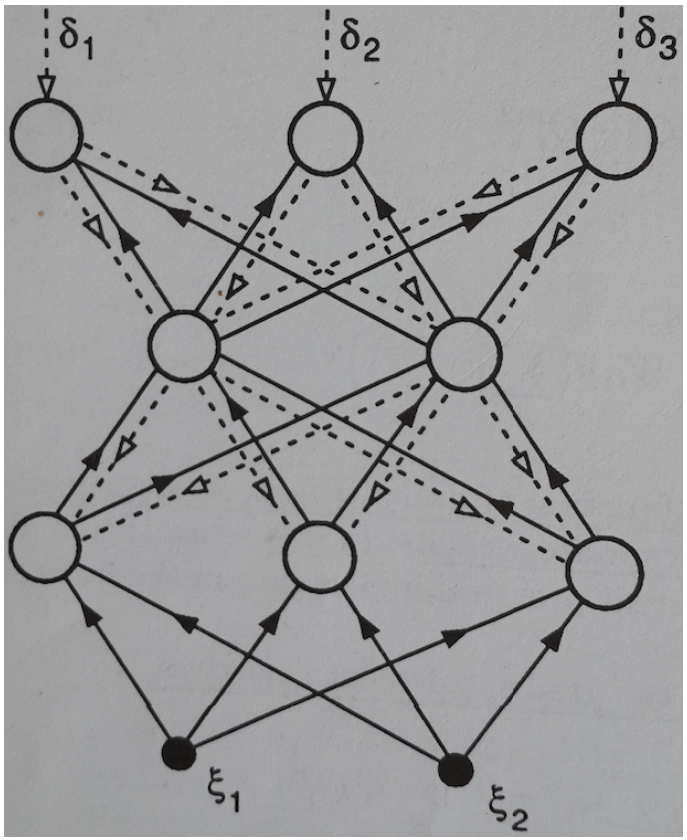


En cada capa, representamos distintos niveles de abstracción.

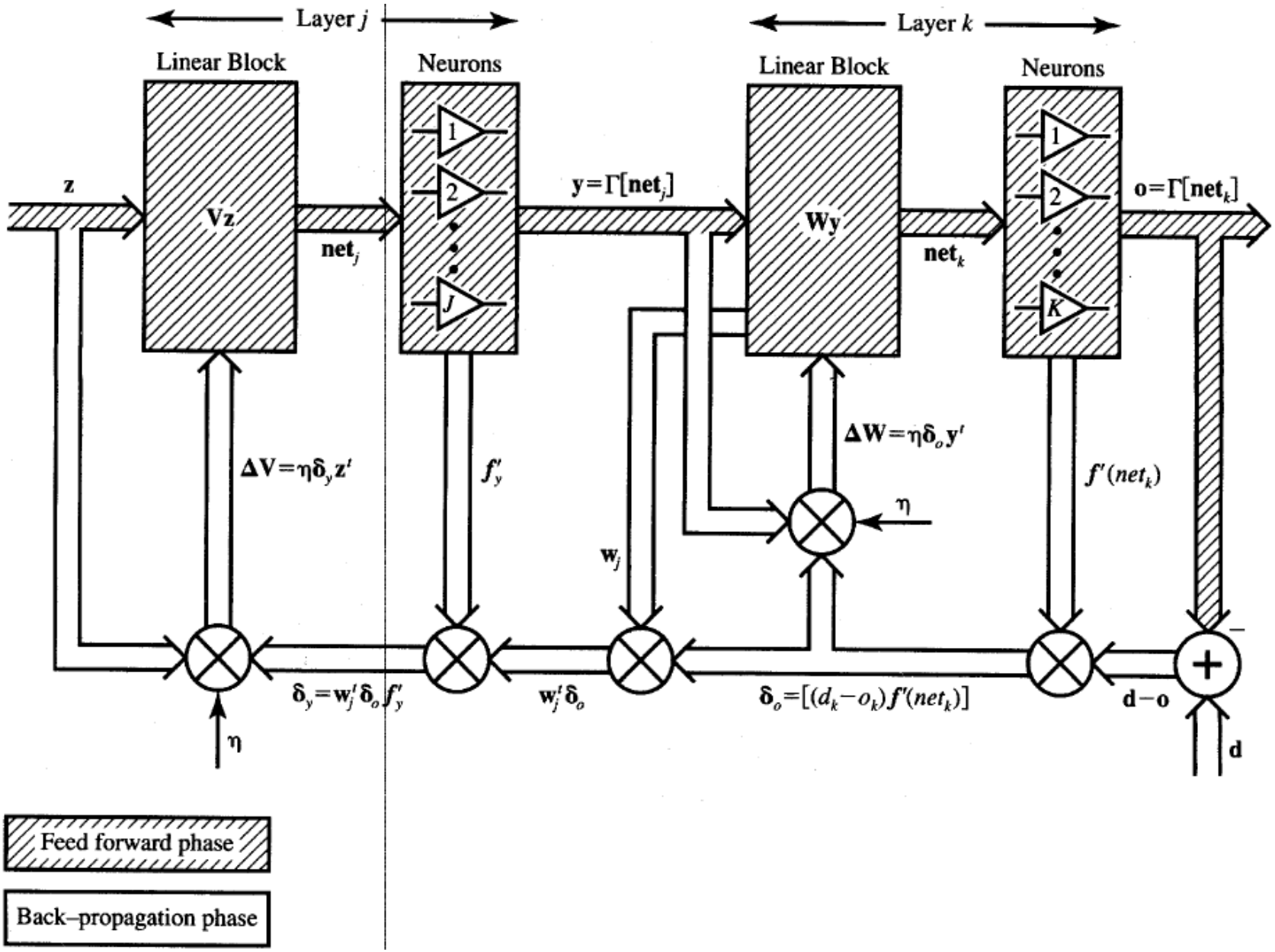
Algoritmo de Backpropagation

La regla de aprendizaje está basada en minimizar la función de costo $E(w)$, siguiendo la dirección del gradiente descendiente.

1. *Etapa forward:* Calcula el output de la red neuronal.
2. *Etapa backward:* Calcula el gradiente descendiente de $E(w)$ con respecto a los pesos de la red, y actualiza los pesos.



Algoritmo de Backpropagation (diagrama)



Datasets

El conjunto de datos, se particiona usualmente en los siguientes conjuntos disjuntos:

1. Training set: Datos que se usan para entrenar la red (son los que ajustan los pesos).
2. Test set: Datos que no participan en el entrenamiento, y se usan luego del entrenamiento, para medir la performance final de la red.
3. Validation set: Datos que no modifican los pesos de la red, pero que se usan durante el entrenamiento, para mejorar la performance de la red, haciendo early-stopping y evitando un sobre-entrenamiento de la red, que podría causar un overfitting.

Entrenamiento: épocas

Se denomina *época* a pasar por la red neuronal todos los patrones del training set (uno a la vez).

Tip: Conviene cambiar en forma random en cada época el orden en el cual se pasan los patrones de entrenamiento, ya que esto acelera el aprendizaje.

Medidas de performance de la red

Usualmente, para evaluar la performance de la red, se usa el Mean Squared Error (MSE):

$$E(w) = \frac{1}{N} \sum_{i\mu} (\zeta_i^\mu - O_i^\mu)^2$$

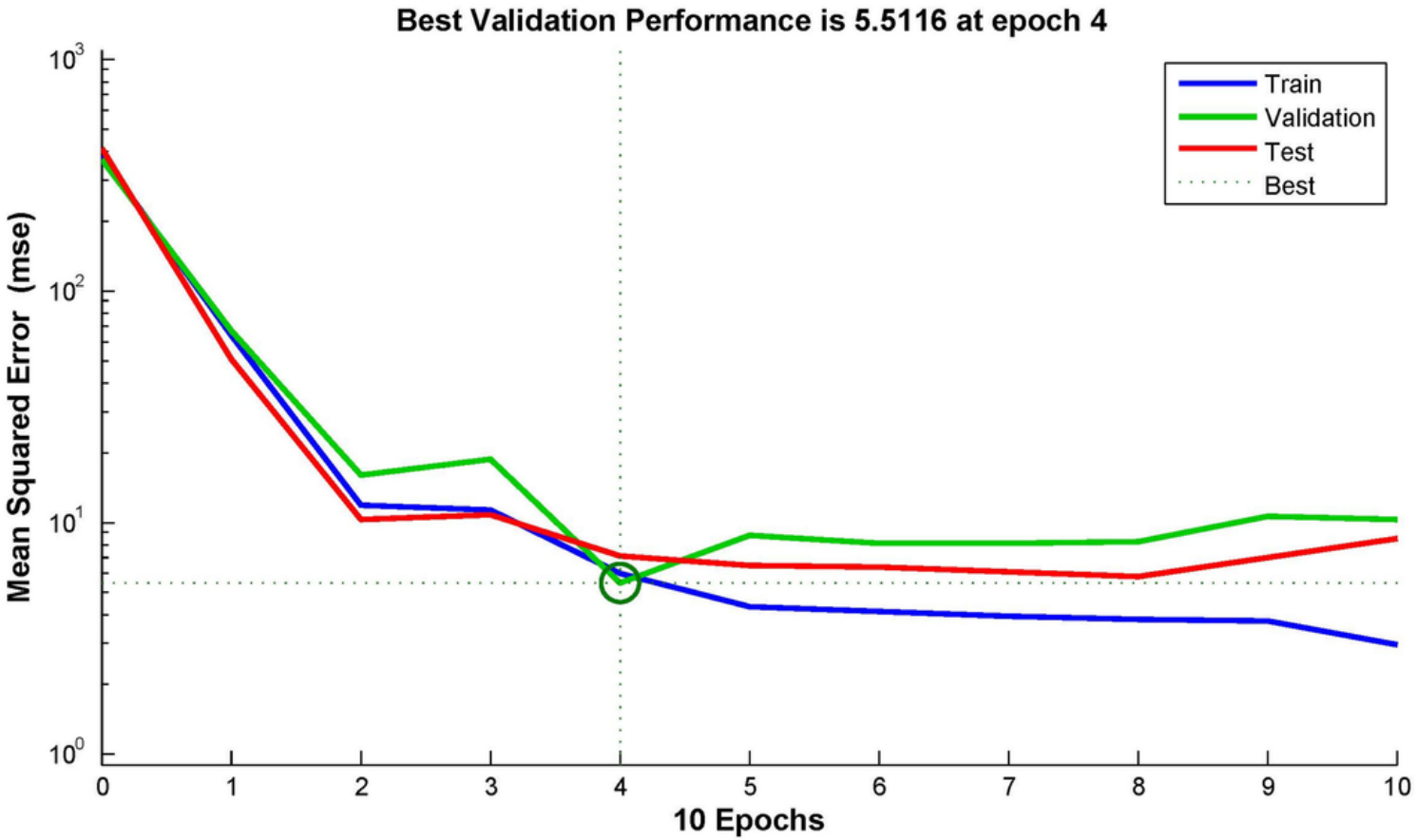
donde ζ es la salida deseada de la neurona i, O es el output de la neurona i, μ es el patrón de entrada, i es la unidad de salida, y N es la cantidad total de patrones del conjunto correspondiente.

Otra medida también comúnmente utilizada es el Root Mean Squared Error (RMSE) que es la raíz cuadrada del MSE.

Procedimiento general de entrenamiento

Durante el entrenamiento, al final de cada época, medimos los MSE (o RMSE) de los conjuntos de entrenamiento y validación. Mientras estos errores sigan disminuyendo, continuamos el entrenamiento. Cuando el MSE del conjunto de validación comienza a ascender, detenemos el entrenamiento (early-stopping) para evitar un overfitting, y medimos el MSE del conjunto de testing. Si los MSE de los conjuntos de training y testing, son lo suficientemente bajos, terminamos. Sino, habrá que conseguir más datos de entrada, o modificar la estructura de la red, ya que puede estar sucediendo que la red no sea lo suficientemente compleja como para modelar el problema.

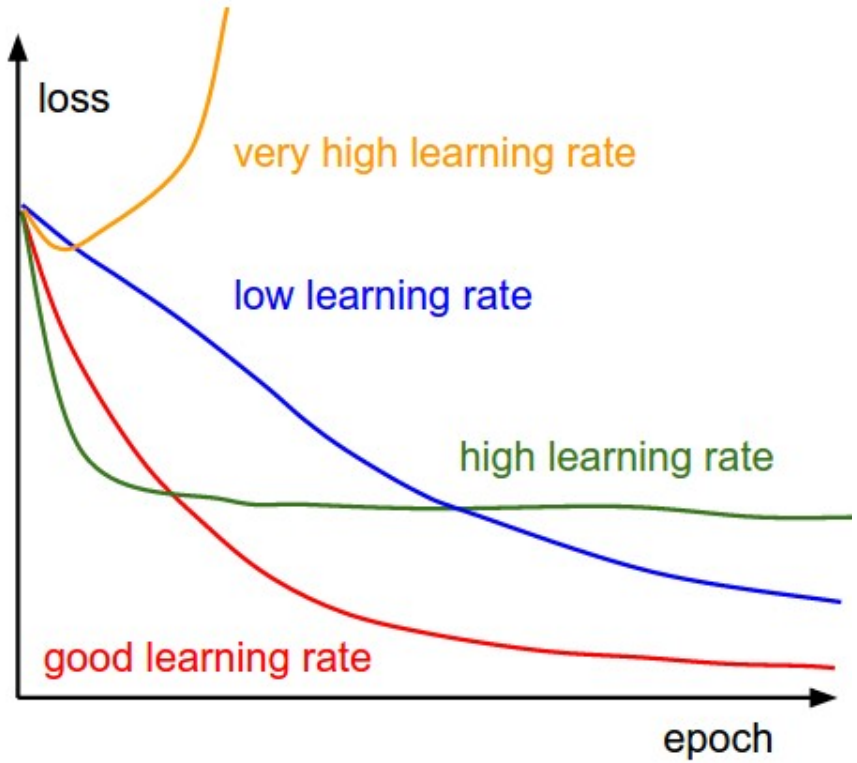
Ejemplo: gráficos de error de los datasets



Pseudocódigo super-simplificado de la red neuronal feedforward multicapa

1) *Inicialización.* Inicializar pesos de la red en forma random.
2) *Entrenamiento.* Mientras MSE > umbral hacer:
2.1) *Epoca:* Mientras haya patrones en conjunto de training:
2.1a) *Activación.* Elegir un patrón del conjunto de training.
2.1b) *Cómputo de la respuesta actual.* Calcular en forma forward el output de la red neuronal.
2.1c) *Cómputo del error.* Calcular el error de la red con respecto a la salida esperada.
2.1d) *Cómputo de los deltas.* Calcular los deltas de la red en forma backward.
2.1e) *Adaptación de los pesos.* Actualizar los pesos de la red.
2.2) *Cálculo MSE:* Calcular el MSE del conjunto de training (opcionalmente, también el MSE del conjunto de validación , si hacen early-stopping).

Factor de aprendizaje: η



El factor de aprendizaje η es el escalar que aparece en la regla Δw de actualización de los pesos, y equivale al *tamaño del paso*, en la dirección de descenso de la función de costo. Si el paso es muy grande, aunque la dirección sea correcta, el aprendizaje diverge. Si el paso es muy pequeño, tardamos demasiado en converger al mínimo.

Laboratorio

Abrir las Jupyter Notebooks en Google Colab:

https://github.com/deep-learning-indaba/indaba-2018