Redes Neuronales Artificiales

Eduardo Morales, Hugo Jair Escalante

Coordinación de Ciencias Computacionales
Instituto Nacional de Astrofísica, Óptica y Electrónica

Septiembre, 2015

Introducción

A las redes neuronales (conneccionismo, proceso paralelo distribuido, computación neuronal, redes adaptivas, computación colectiva) las podemos entender desde dos puntos de vista:

- **Computacional:** representar funciones usando redes de elementos con cálculo aritmético sencillo, y métodos para aprender esa representación a partir de ejemplos. La repesentación es útil para funciones complejas con salidas continuas y datos con ruido
- **Biológico:** modelo matemático de la operación del cerebro. Los elementos sencillos de cómputo corresponden a neuronas, y la red a una colección de éstas.

Introducción

- La neurona es la unidad funcional fundamental del sistema nervioso.
- Cada neurona tiene un cuerpo (soma) que tiene un nucleo y tiene un grupo de fibras (dendritas) y una de las cuales es más larga (axón).

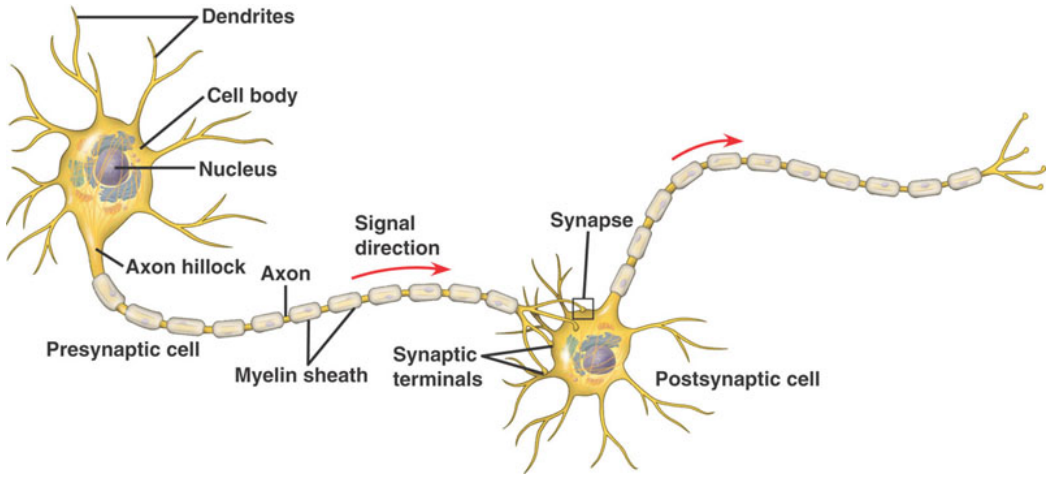


Figura: Neurona.

Introducción

- El axón se bifurca eventualmente en sinapses. Las señales se propagan en una reacción electroquímica complicada.
- Las substancias químicas transmisoras se liberan de las sinapses y entran a la dendrita, aumentando o disminuyendo el potencial eléctrico del cuerpo de la célula.

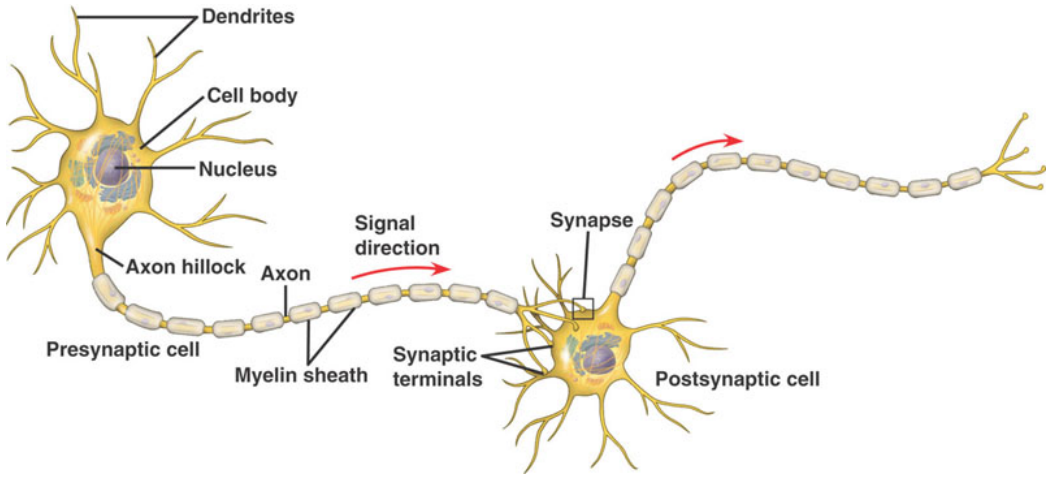
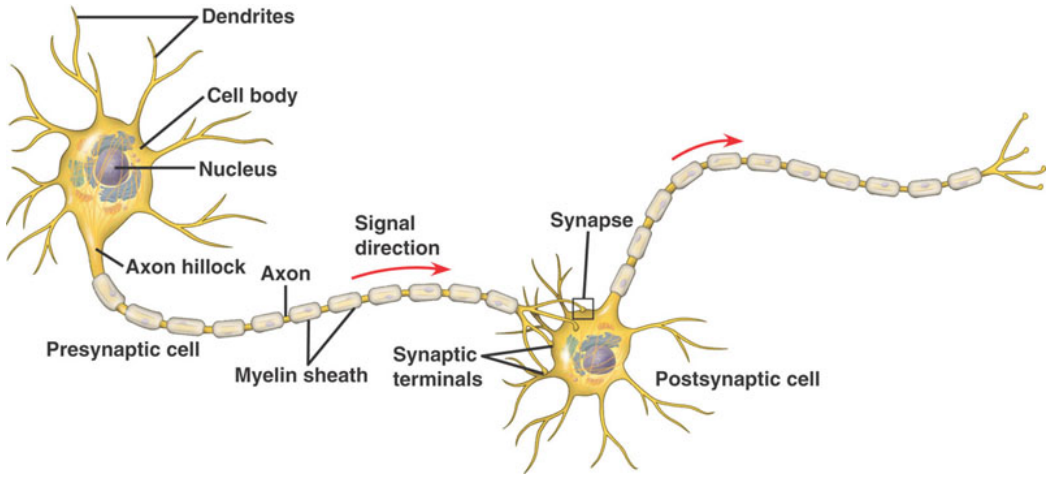


Figura: Neurona.

Introducción

- Cuando el potencial alcanza un umbral se transmite un pulso elétrico o acción potencial a través del axón. Las sinapses que aumentan el potencial se llaman exitatorias y los que disminuyen, inhibidoras.
- La conexión “sináptica” es *plástica* (cambia con la estimulación).
- Se pueden formar nuevas conecciones y las neuronas migran de un lugar a otro. Esto se cree que forman la base de aprendizaje en el cerebro.



Introducción

- En general el mapeo de regiones con funciones puede ser múltiple y cambiar cuando un área es dañada (pero no se sabe bien como se hace).
- Lo sorprendente es que una colección de células simples puedan dar pensamiento, acción y conciencia (*cerebros causan mentes* (Searle 92)).

Introducción

Cuadro: Comparación gruesa de las capacidades computacionales de cerebros y computadoras (1994).

	Computadora	**Cerebro Humano**
Unidades Computacionales	1 CPU, 10^5 compuertas	10^{11} neuronas
Unidades de Almacenamiento	10^9 bits RAM, 10^{10} bits disco	10^{11} neuronas, 10^{14} sinapses
Ciclo (tiempo)	10^{-8} seg.	10^{-3} seg.
Anchobanda	10^9 bits/seg.	10^{14} bits/seg.
Actualizaciónes/seg.	10^5	10^{14}

A pesar de que una computadora es millones de veces más rápida por proceso individual, el cerebro finalmente es billones de veces más rápido (ver tabla 1).

Introducción

- Una de las atracciones, es construir un mecanismo que combine el paralelismo del cerebro con la velocidad de las máquinas.
- Los cerebros son mucho más tolerantes (en 70-80 años, no se tiene que reemplazar una tarjeta de memoria, llamar al servicio o hacer reboot).
- La tercera atracción es su degradación gradual.

Historia

- Existió mucho desarrollo en los primeros años de la computación: McCulloch y Pitts (43), Hebb (49), Minsky (51) (primera red), Ashby (52), Rosenblatt (57) (perceptrón), Selfridge (59) (pandemonium), Widrow y Hoff (60) (adalines), Nilsson (65 - 90), Minsky y Papert (69).
- Durante 10 años prácticamente no se hizo nada.
- El resurgimiento comenzo en la decada de los 80's: Hinton y Anderson (81), Hopfield (82), Hinton y Sejnowski (83 y 86) y los dos volumens de PDP (Parallel Distributed Processing) anthology (Rumelhart *et al.* 86).

Historia (reciente)

- Durante los 95's - 03's hubo otra época de oscurantismo en RNs, debido al surgimiento y popularización de SVM.
- Las RNs tuvieron (otro) *segundo aire* a finales de la primera decada del presente siglo.

Redes neuronales artificiales

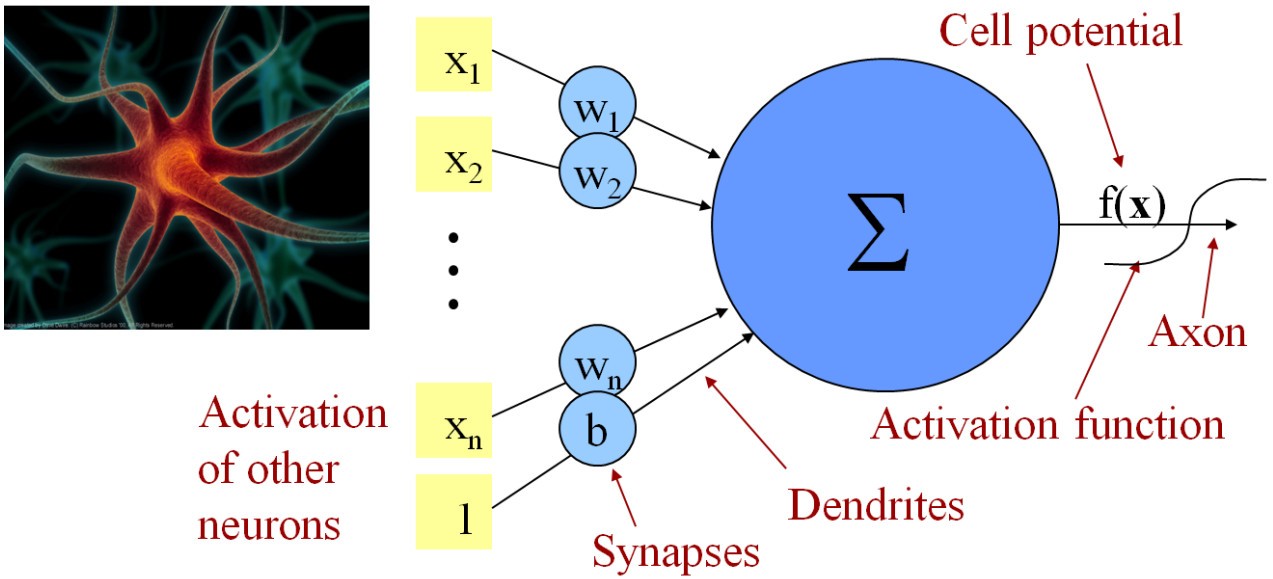
- El funcionamiento de las neuronas y del cerebro en general sirve como *insipración* para el desarrollo de sistemas de aprendizaje computational.
- El equivalente computacional de una neurona es una *unidad* que almacena pesos asociados a un problema de aprendizaje.
- Redes de neuronas, imitan de manera burda el funcionamiento del cerebro.

Redes neuronales artificiales

En pocas palabras una RNA es:

- Un modelo no-lineal, formado por muchos modelos (unidades) lineales con funciones de activación no-lineal.
- Un modelo que modifica los valores de sus elementos para hacer corresponder sus salidas con las salidas esperadas/verdaderas.

Redes neuronales artificiales



McCulloch and Pitts, 1943

$$f(\mathbf{x}) = \mathbf{w} \bullet \mathbf{x} + b$$

Figura: Neurona artificial (diap. I. Guyon.).

Redes neuronales artificiales

- Una red neuronal está compuesta por nodos o unidades, conectados por ligas. Cada liga tiene un peso numérico asociado. Los pesos son el medio principal para almacenamiento a largo plazo en una red neuronal, y el aprendizaje normalmente se hace sobre la actualización de pesos.

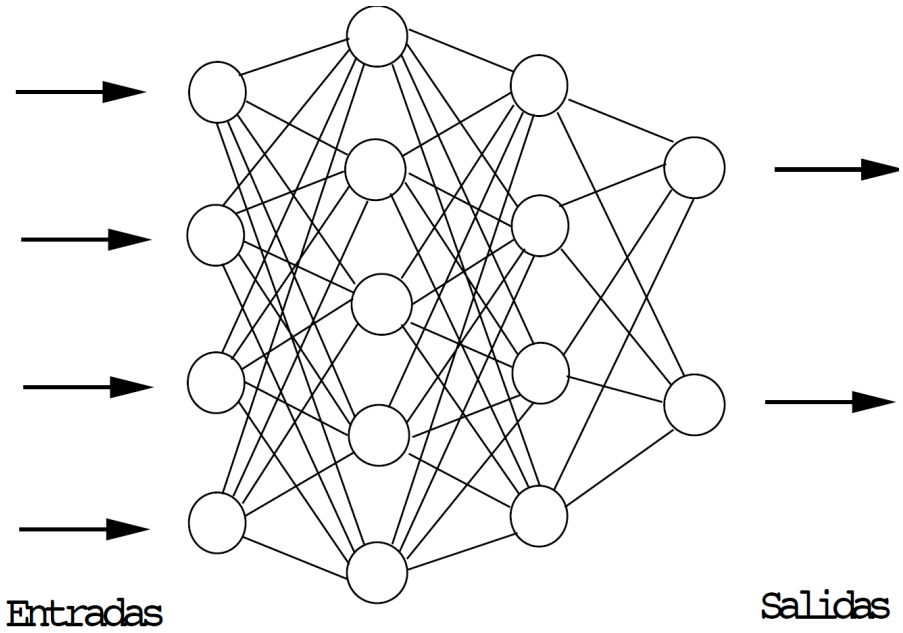


Figura: Red Neuronal prototípica.

Redes neuronales artificiales

- Algunas unidades están conectadas al medio ambiente externo y pueden diseñarse como unidades de entrada o salida.
- Los pesos se modifican para tratar de hacer que el comportamiento entrada/salida se comporte como el del ambiente.

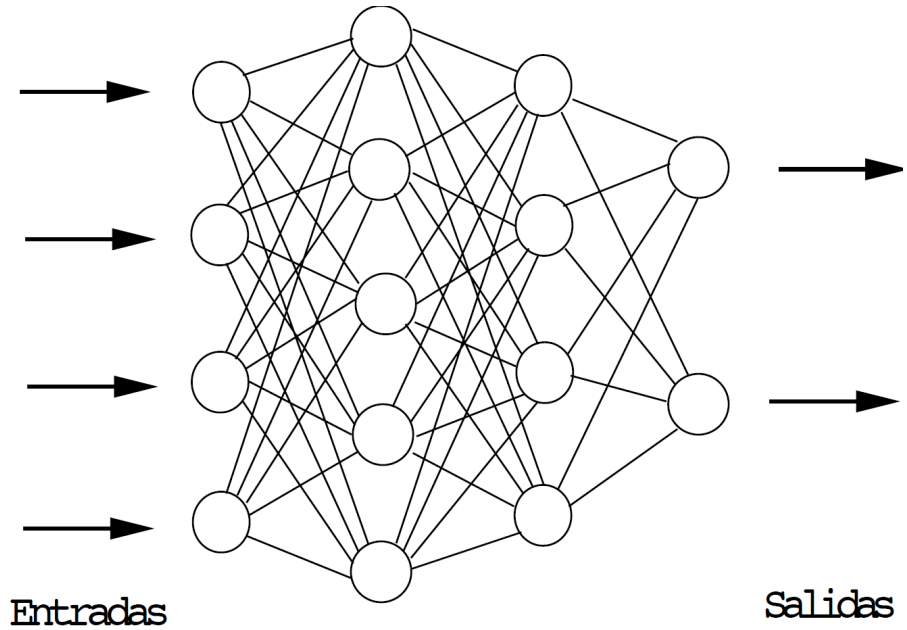


Figura: Red Neuronal prototípica.

Redes neuronales artificiales

- Cada unidad tiene un conjunto de ligas de entrada (provenientes de otras unidades) y un conjunto de ligas de salida (hacia otras unidades), un nivel de activación, y una forma de calcular su nivel de activación en el siguiente paso en el tiempo, dada su entrada y sus pesos (cada unidad hace un cálculo local basado en las entradas de sus vecinos).

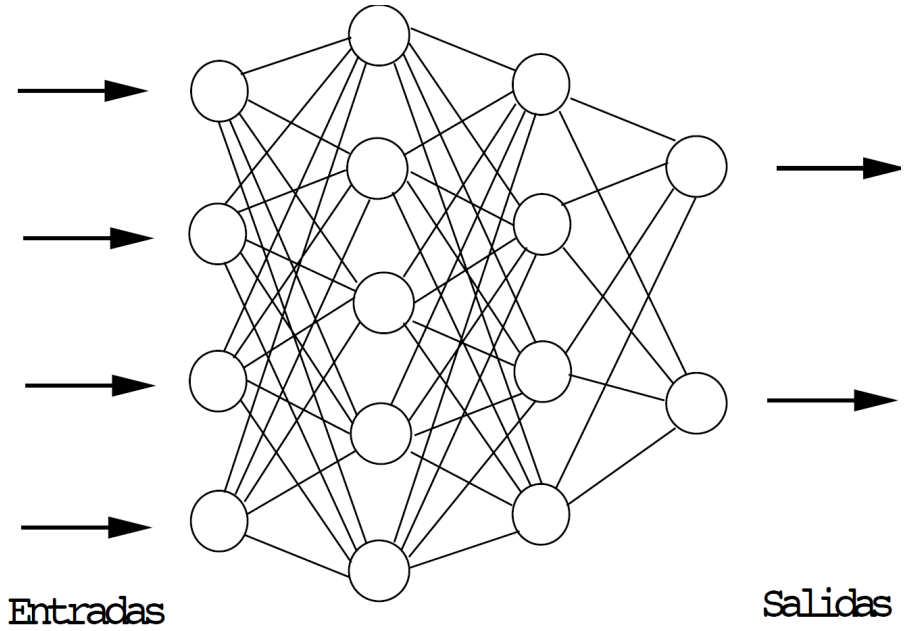


Figura: Red Neuronal prototípica.

Redes neuronales artificiales

La computación se hace en función de los valores recibidos y de los pesos.
Se divide en dos:

1. Un componente lineal, llamado la función de entrada (in_i), que calcúla la suma de los valores de entrada.
2. Un componente no lineal, llamado función de activación (g), que transforma la suma pesada en una valor final que sirve como su valor de activación (a_i).

Normalmente, todas las unidades usan la misma función de activación.

Redes neuronales artificiales

La suma pesada es simplemente las entradas de activación por sus pesos correspondientes:

$$in_i = \sum_j w_{j,i} a_j = \mathbf{w_i} \cdot \mathbf{a_i}$$

$\mathbf{w_i}$: vector de los pesos que llegan a la unidad i
$\mathbf{a_i}$: vector de los valores de activación de las entradas a la unidad i
El nuevo valor de activación se realiza aplicando una función de activación g:

$$a_i \leftarrow g(in_i) = g(\sum_j w_{j,i} a_j)$$

Se obtienen modelos diferentes cambiando g. Las opciones comunes son (ver figura 17):

- función escalón:

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

- signo:

$$signo(x) = \begin{cases} +1, & \text{si } x \geq 0 \\ -1, & \text{si } x < 0 \end{cases}$$

- sigmoide:

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

Se obtienen modelos diferentes cambiando g. Las opciones comunes son (ver figura 17):

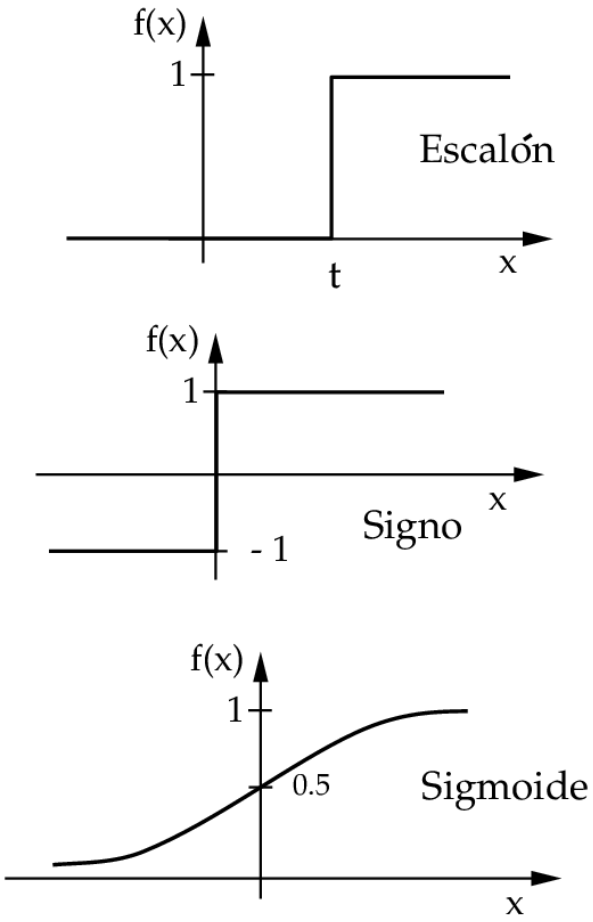


Figura: Funciones de activación comunes para Redes Neuronales.

Redes neuronales artificiales

- En la práctica, casi todas las implementaciones de RN son en software y utilizan un control síncrono en su actualización.

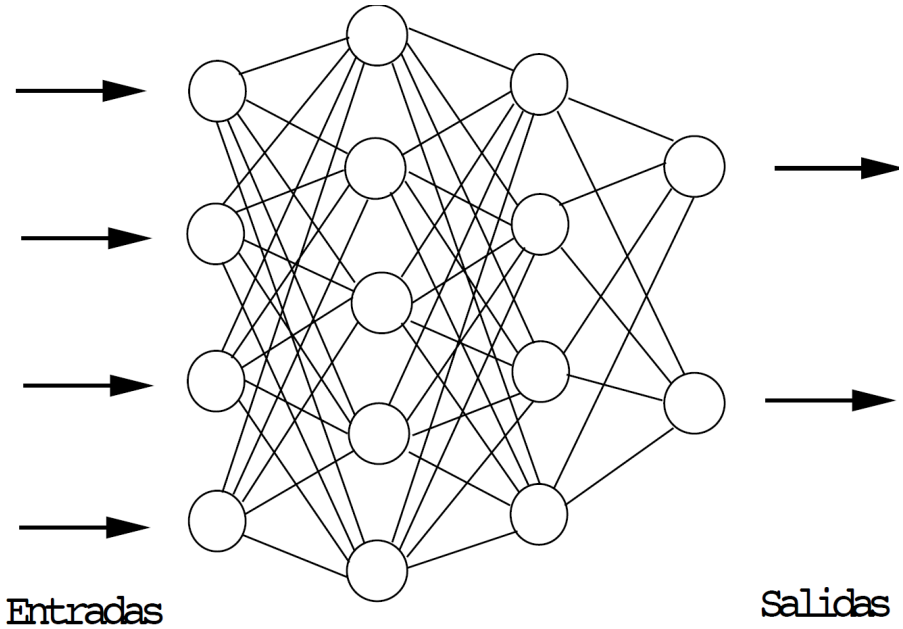


Figura: Red Neuronal prototípica.

Redes neuronales artificiales

Ejemplo de aplicación:

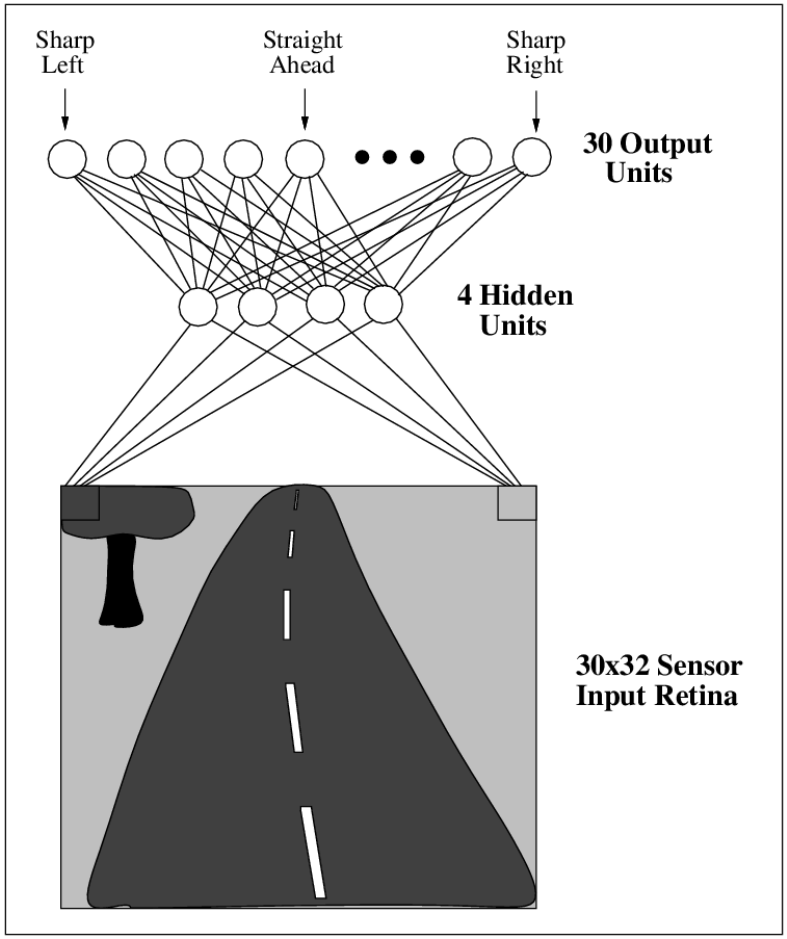


Figura: Arquitectura de ALVINN.

Redes neuronales artificiales

Para el diseño uno debe de decidir:

- número de unidades
- cómo se deben de conectar
- qué algoritmo de aprendizaje utilizar
- cómo codificar los ejemplos de entradas y salidas

Cada unidad recibe señales de sus ligas de entradas y calcúla un nuevo nivel de activación que manda a través de sus ligas de salidas.

Redes neuronales artificiales

Problemas apropiados para abordarse con RNA:

- Las instancias se representan por muchos pares de atributo-valor.
- La función objetivo de salida puede ser discreta, real, un vector de reales-categorías o una combinación de ambos.
- Los ejemplos de entrenamiento pueden tener errores.
- Se requiere una evaluación rápida de la función aprendida.
- No es importante interpretar la función aprendida.

Estructuras de redes

Existen muchas estructuras que dan propiedades computacionales distintas.
La distinción principal es entre:

1. *feed-forward*: ligas unidireccionales, sin ciclos (DAGs). Normalmente estaremos hablando de redes que están arregladas en capas. Cada unidad está ligada solo con las unidades de la siguiente capa. No hay ligas inter-capas, ni ligas a capas anteriores, ni ligas saltandose capas.
2. *recurrent*: las ligas pueden formar topologías arbitrarias.

Estructuras de redes

- Una red *feed-forward* calcula una función de las entradas que depende de los pesos. Este es el modelo más usado y nos vamos a concentrar más en éste.
- Por un lado, están las unidades de entrada (su valor de activación depende del medio ambiente). Del otro, las unidades de salida. En medio (sin conección al medio ambiente) se tienen las unidades ocultas (ver figura 11).

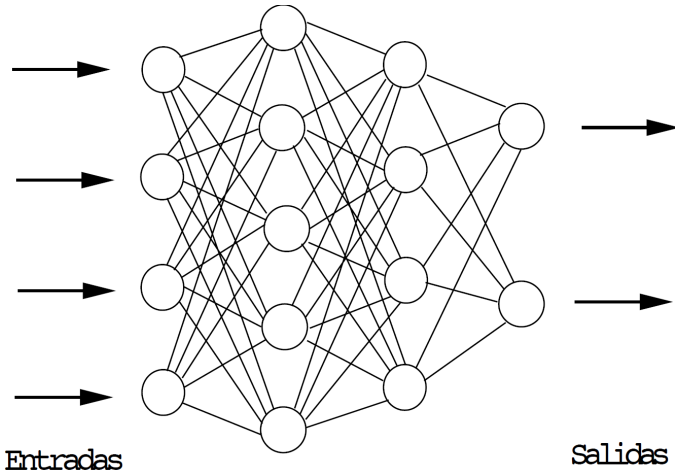


Figura: Arquitectura típica de una Red Neuronal *feedforward*.

Estructuras de redes

- Algunas redes no tienen nodos o unidades ocultos (*perceptrones*). Esto hace el aprendizaje mucho más sencillo, pero limita lo que se puede aprender.
- Redes con una o mas capas ocultas se llaman redes multicapas.

Estructuras de redes

- Con una sola capa (suficientemente grande) de unidades ocultas, es posible representar cualquier función continua de las entradas. Con dos capas es posible representar hasta funciones discontinuas.
- Con una estructura fija y función de activación *g* fija, las funciones representables por una red *feed-forward* están restringidas por una estructura específica parametrizada.

Estructuras de redes

Los pesos escogidos para la red determinan cuáles de las funciones se representan.
Por ejemplo, una red con 2 unidades de entrada, dos ocultas y una de salida, con todas las conecciones intercapas, calcula la siguiente función (ver figura 12):

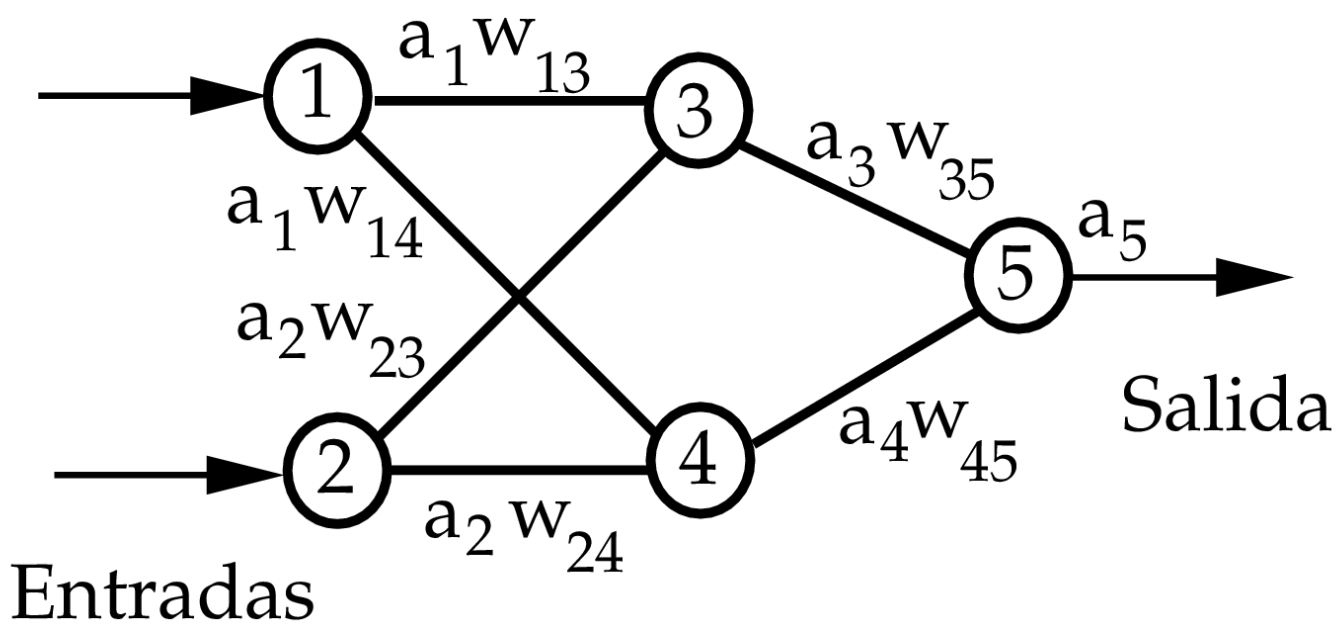


Figura: Arquitectura de una Red Neuronal simple.

Estructuras de redes

$$a_5 = g(w_{3,5}a_3 + w_{4,5}a_4)$$

$$= g(w_{3,5}g(w_{1,3}a_1 + w_{2,3}a_2) + w_{4,5}g(w_{1,4}a_1 + w_{2,4}a_2))$$

Como g es una función no lineal, la red representa una función nolineal compleja.
Si se piensa que los pesos son los parámetros o coeficientes de esta función, el aprendizaje es simplemente el proceso de "afinar" los parámetros para que concuerden con los datos en el conjunto de entrenamiento (es lo que en estadística se llama regresión nolineal).

El Perceptron

Un tipo de *unidad* de aprendizaje en RNs es el perceptron: Clasificador lineal (red con una única neurona!) que se basa en un vector de pesos ($\mathbf{w}$), tal que, dado un ejemplo, determina si éste pertenece a la clase positiva ($+1$) o negativa (-1).

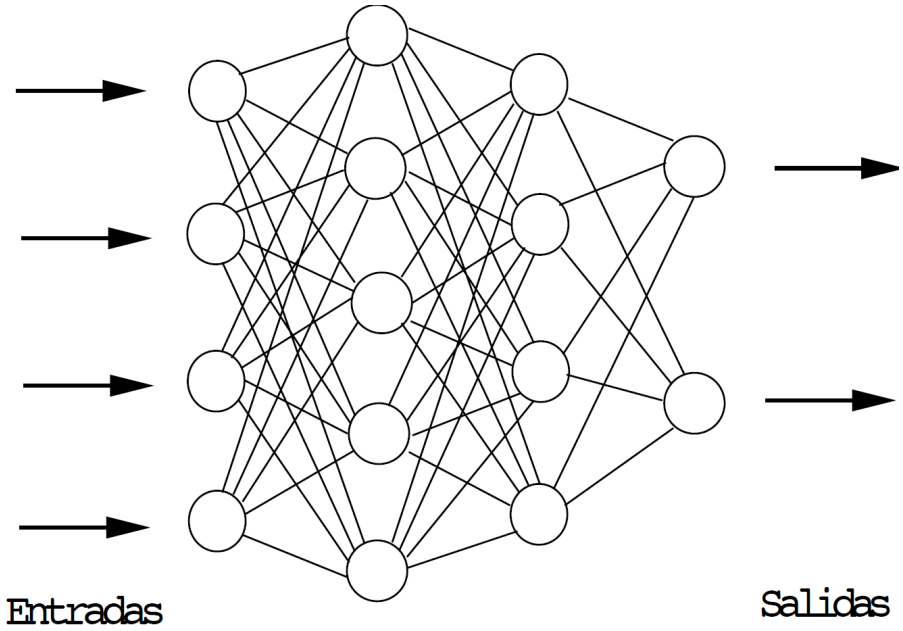


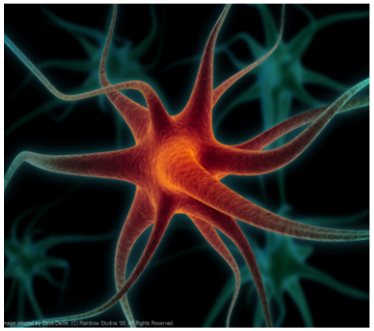
Figura: Red Neuronal prototípica.

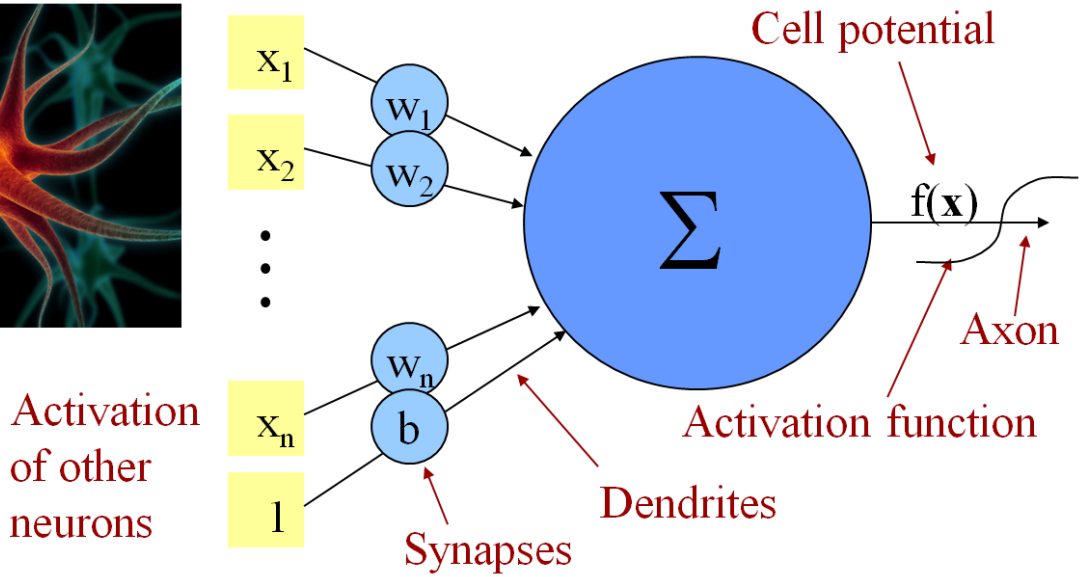
El Perceptron

Dado un ejemplo, determina si éste pertenece a la clase positiva $(+1)$ o negativa (-1):

$$f(x) = sign(wx + b)$$

Redes neuronales artificiales



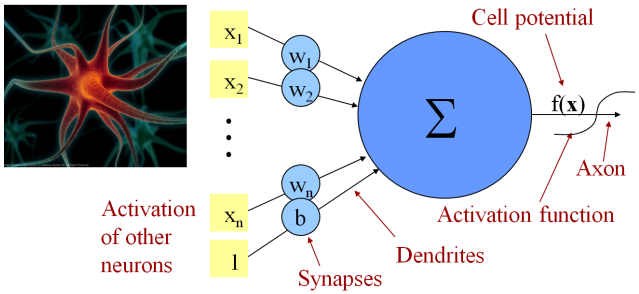


McCulloch and Pitts, 1943

$$f(\mathbf{x}) = \mathbf{w} \bullet \mathbf{x} + b$$

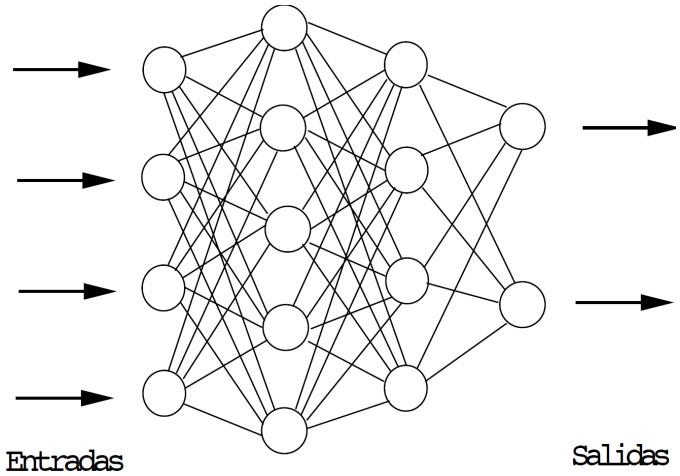
Figura: Neurona artificial (diap. I. Guyon.).

Redes neuronales artificiales



McCulloch and Pitts, 1943

$$f(\mathbf{x}) = \mathbf{w} \bullet \mathbf{x} + b$$



El Perceptron

Cómo determinar los pesos w?

- Una forma de hacerlo es iniciar con w generado aleatoriamente e iterativamente aplicar el perceptron a cada ejemplo de entrenamiento, modificando w cada vez que un ejemplo sea mal classificado.
- Este procedimiento se itera hasta que el perceptron clasifique todos los ejemplos correctamente.

Los pesos se modifican en cada paso mediante la regla de entrenamiento del perceptron:

$$w_i \leftarrow w_i + \Delta w_i$$

donde:

$$\Delta w_i = \eta(t - o)x_i$$

y
$t \in -1, 1$ es el valor real de $f(x)$, o es la salida generada por el perceptron, y η es la tasa de aprendizaje.

El Perceptron

Cuadro: Algoritmo Aprendizaje del Perceptron

$w \leftarrow$ pesos asignados aleatoriamente
repeat
para cada $x \in$ *ejemplos* do
    $o \leftarrow wx + b$ // salida del perceptron
    $t \leftarrow$ valor observado de x // i.e., $f(x)$
    $\Delta w = \eta(t - o)x$ // Calcula Δw
    $w \leftarrow w + \Delta w$ // Actualiza w
    end
until todos los ejemplos sea predichos correctamente o se alcance un criterio de paro
regresa w

El Perceptron

- Se ha demostrado que el procedimiento anterior converge después de un número finito de iteraciones a un vector de pesos que separa todos los datos de entrenamiento: *siempre y cuando sea un problema separable linealmente.*
- Qué pasa si el problema no es linealmente separable?

Regla delta

Una regla de aprendizaje similar a la del perceptron, que puede lidiar con problemas que no son linealmente separables.
La idea fundamental es usar gradiente descendente para encontrar el vector de pesos que mejor se ajuste a los ejemplos de entrenamiento (la base para *backpropagation*).

Regla delta

La regla delta se entiende mejor cuando la tarea es aprender un perceptron sin umbral:

$$o(x) = wx$$

Con $w =< w_0, w_1 >$, $x \leftarrow < 1, x >$
Queremos minimizar el error:

$$E(w) = \frac{1}{2} \sum_{i=1}^{N} (t_i - o_i)^2$$

Solución: usar gradiente descendente!

Cómo aprender los pesos?

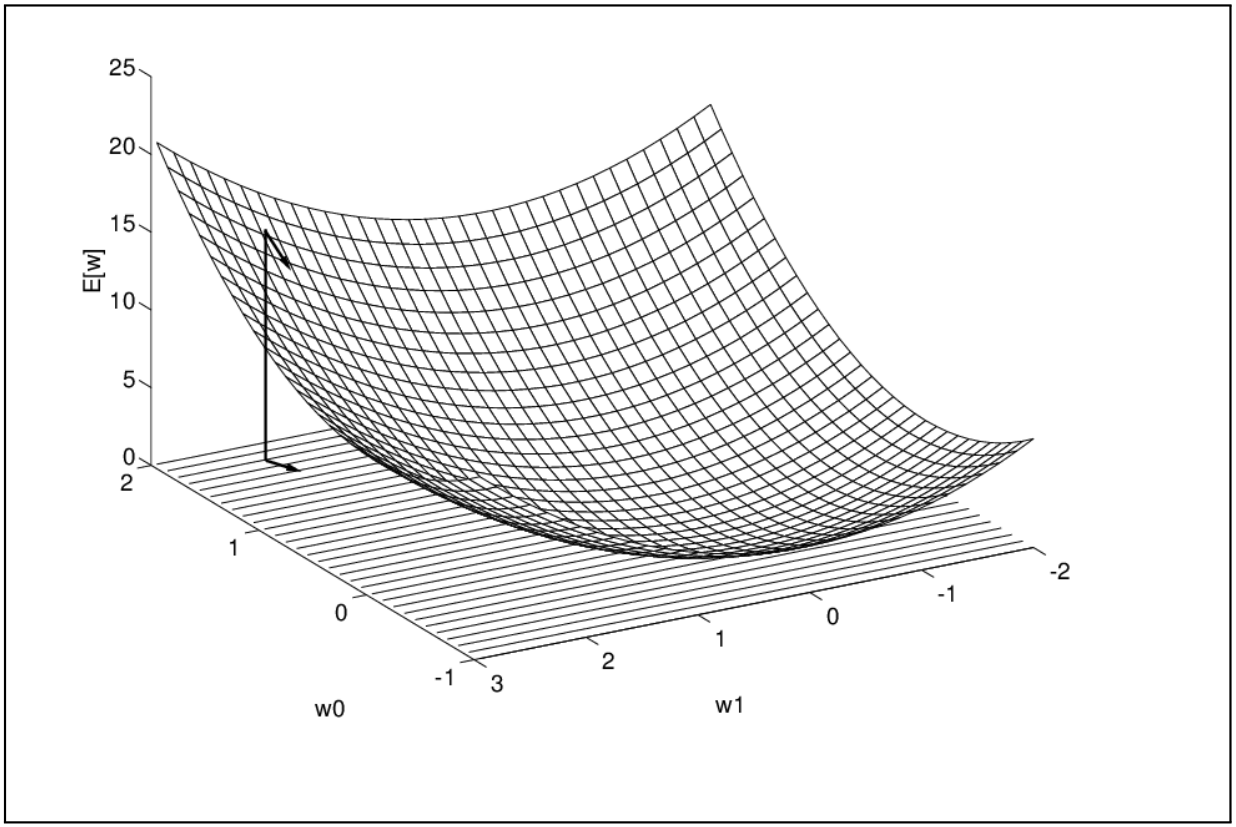


Figura: Error para diferentes hipótesis.

Gradiente descendiente y la regla Delta

Lo que queremos es determinar el vector de pesos que minimice el error E.

$$E(W) = \frac{1}{2}\sum_i (t_i - o_i)^2$$

Esto se logra alterando los pesos en la dirección que produce el máximo descenso en la superficie del error.
La dirección de cambio se obtiene mediante el gradiente. El gradiente nos especifica la dirección que produce el máximo incremento, por lo que el mayor descenso es el negativo de la dirección.

La regla de actualización de pesos es entonces:

$$W \leftarrow W + \Delta W$$

$$\Delta W = -\alpha \nabla E$$

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\ &= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\ &= \sum_{d \in D} (t_d - o_d)(-x_{i,d}) \end{aligned}$$

Por lo que:

$$\Delta w_i = \alpha \sum_{d \in D} (t_d - o_d) x_{i,d}$$

Regla delta

Cuadro: Algoritmo Aprendizaje, regla delta

$w \leftarrow$ pesos asignados aleatoriamente
repeat
Incializa cada Δw_i en cero.
Para cada $x \in$ *ejemplos* do
    $o \leftarrow wx + b$ // salida de la unidad lineal
    $t \leftarrow$ valor observado de x, i.e., $f(x)$
    Para cada peso de la unidad lineal do
        $\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$ // Calcula Δw
    end
end
$w \leftarrow w + \Delta w$ // Actualiza w
until se alcance un criterio de paro
regresa w

En la práctica, se tiende a usar un gradiente descendiente estocástico. Esto es, en lugar de procesar el error sobre todos los datos, se hace sobre uno solo.
En este caso, la regla de actualización es:

$$\Delta w_i = \alpha(t - o)x_i$$

- Rosenblatt y otros se concentraron en una sola capa, por no encontrar un método adecuado de actualizar los pesos entre las entradas y las unidades ocultas, cuando el error se calcula en las unidades de salida.
- Minsky y Papert dijeron que investigar multicapas era un problema de importancia, pero especularon que no había razón para suponer que alguna de las virtudes de los perceptrones (teorema de regla de aprendizaje) se mantuvieran con multicapas y que su extensión sería esteril.
- En parte tuvieron razón, pero definitivamente no ha sido esteril. Aprendizaje en multicapas no es eficiente ni garantiza converger al óptimo global. El aprender funciones generales a partir de ejemplos es un problema intratable en el peor de los casos.

Redes multi-capa y retro-propagación

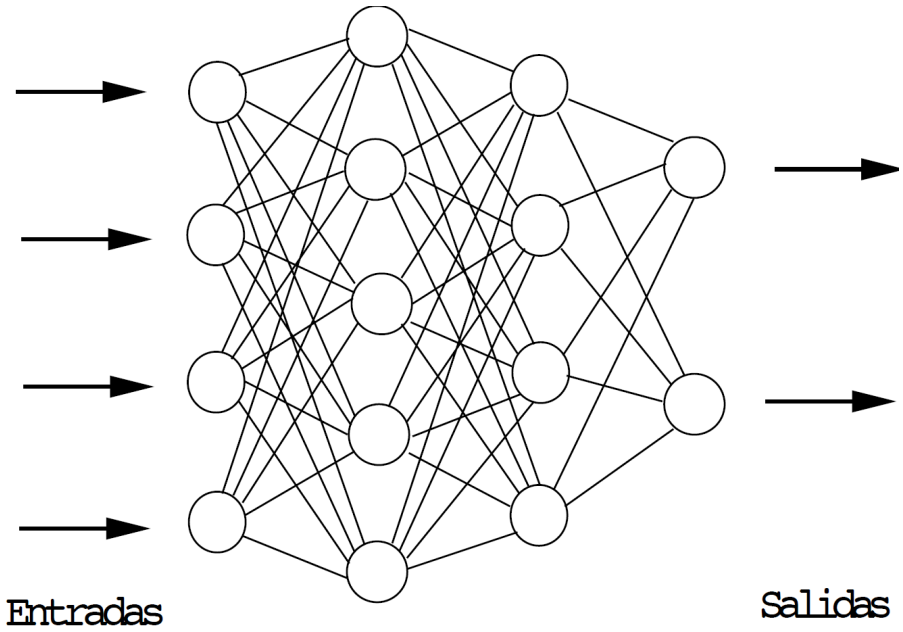


Figura: Red Neuronal prototípica.

Redes multi-capa y retro-propagación

- Redes multicapa con unidades lineales producen salidas lineales.
- Idea: introducir función de activación no lineal sobre la salidas de unidades lineales.
- Si las funciones son diferenciables, se pueden derivar métodos de aprendizaje basados en GD/SGD.

Redes multi-capa y retro-propagación

- función escalón:

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

- signo:

$$signo(x) = \begin{cases} +1, & \text{si } x \geq 0 \\ -1, & \text{si } x < 0 \end{cases}$$

- sigmoide:

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

Redes multi-capa y retro-propagación

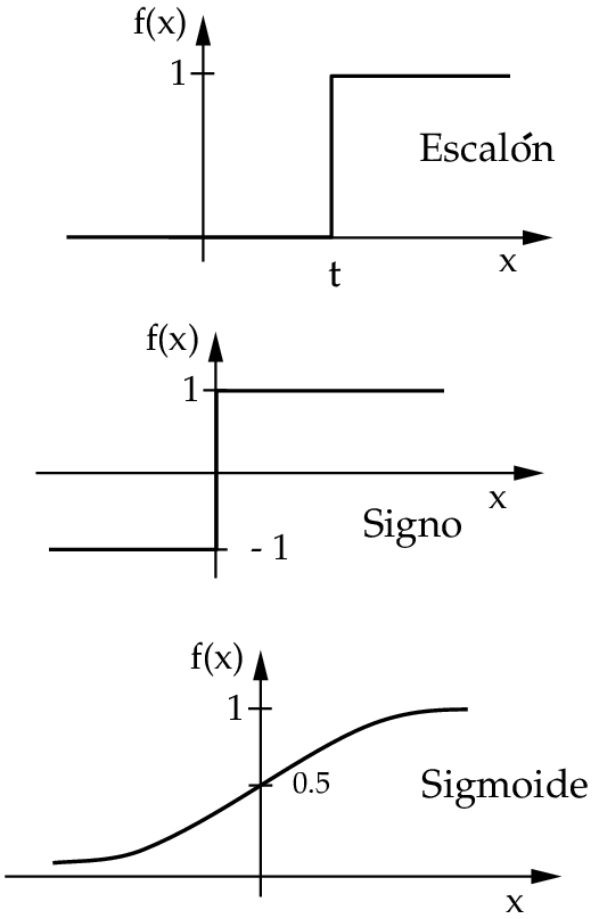


Figura: Funciones de activación comunes para Redes Neuronales.

Algoritmo de retropropagación

El método más popular para aprender redes multicapas es el de retro-propagación (*back-propagation*).
Se publicó originalmente en 1969 por Bryson y Ho, pero fué ignorado hasta mediados de los 80's.
Aprender en una red multicapas es muy parecido a un perceptrón. Si existe un error se ajustan los pesos para reducir el error.
El truco es dividir la *culpa* del error entre los pesos contribuyentes. Como en el perceptrón se trata de minimizar el error (en este caso, el cuadrado del error).

Algoritmo de retropropagación

En la capa de salida, la actualización es muy parecida a la de regla delta. Las diferencias son:

- se usa la activación de la unidad oculta a_i en lugar de la de entrada
- la regla contiene un término para el gradiente de la función de activación

Redes multi-capa y retro-propagación

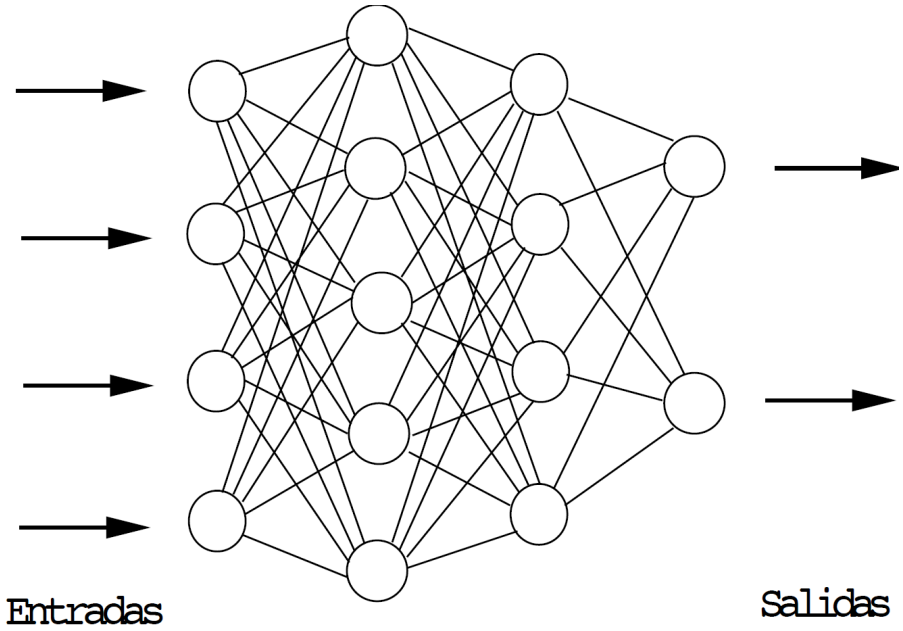


Figura: Red Neuronal prototípica.

Notación:

- x_{ij} = la i-ésima entrada al nodo j
- w_{ij} = el peso asociado a la i-ésima entrada del nodo j
- $net_j = \sum_i w_{ij} x_{ij}$ (suma pesada de entradas al nodo j)
- o_j = la salida del nodo j
- t_j = la salida esperada del nodo j
- σ = función sigmoide
- sal = el conjunto de nodos de salida
- α = razón de aprendizaje.
- $sal(j)$ = conjunto de nodos cuyas entradas directas incluyen la salida del nodo j

Algoritmo de Retropropagación
(un solo paso un solo ejemplo)

1. Propaga las entradas a través de la red y calcula la salida
2. Propaga el error hacia atrás
   1. para cada unidad de salida k, calcula su error δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

   2. Para cada unidad oculta h, calcula su error δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in sal(h)} w_{hk}\delta_k$$

   3. Actualiza los pesos w_{ij}

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij} \qquad \text{donde} \qquad \Delta w_{ij} = \alpha \delta_j x_{ij}$$

Algoritmo de retropropagación

Desarrollo:

Lo que queremos calcular es la actualizacón de los pesos w_{ij} sumandole Δw_{ij}

$$\Delta w_{ij} = \alpha \frac{\partial E_d}{\partial w_{ij}}$$

$$\frac{\partial E_d}{\partial w_{ij}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

$$= \frac{\partial E_d}{\partial net_j} x_{ij} = \delta_j x_{ij}$$

Capa de salida

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k \in sal}(t_k - o_k)^2$$

La derivada es cero en todos los casos, excepto cuando $k = j$, por lo que:

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2$$

$$= -(t_j - o_j)$$

Capa de salida

Como $o_j = \sigma(net_j)$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j}$$

que es la derivada de la sigmoide:

$$= \sigma(net_j)(1 - \sigma(net_j)) = o_j(1 - o_j)$$

Por lo que:

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)o_j(1 - o_j)$$

y finalmente:

$$\Delta w_{ij} = -\alpha \frac{\partial E_d}{\partial w_{ij}} = \alpha(t_j - o_j)o_j(1 - o_j)x_{ij}$$

Capa oculta

si j es un nodo oculto, ahora en la regla de actualización del peso w_{ij} se debe de considerar las formas indirectas en las que pudo contribuir al error (de alguna forma estamos distribuimos el error), por lo que consideramos todos los nodos a los cuales les llega la salida del nodo oculto j.
Vamos a denotar: $\delta_i = -\frac{\partial E_d}{\partial net_i}$

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in sal(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j}$$

$$\delta_j = \sum_{k \in sal(j)} -\delta_k \frac{\partial net_k}{\partial net_j}$$

$$\delta_j = \sum_{k \in sal(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

Capa oculta

$\frac{\partial net_k}{\partial o_j}$ es diferente de cero, sólo cuando tenemos el término $w_{jk} \cdot x_{jk}$ (donde $x_{jk} = o_j$) en la sumatoria, por lo que:

$$\delta_j = \sum_{k \in sal(j)} -\delta_k w_{jk} \frac{\partial o_j}{\partial net_j}$$

$$\delta_j = \sum_{k \in sal(j)} -\delta_k w_{jk} o_j (1 - o_j)$$

$$\delta_j = o_j (1 - o_j) \sum_{k \in sal(j)} -\delta_k w_{jk}$$

Lo que corresponde a la fórmula del inciso 2(b). Finalmente:

$$\Delta w_{ij} = \alpha \delta_j x_{ij}$$

Algoritmo de retropropagación

La retro-propagación puede ser visto como búsqueda de gradiente descendente en la superficie del error.
La retro-propagación nos da una forma de dividir el cálculo del gradiente entre las unidades, con lo que el cambio en cada peso puede calcularse por la unidad al cual el peso está ligado, usando sólo información local.
Como cualquier gradiente descendiente tiene problemas de eficiencia y convergencia, sin embargo, es un paso para pensar en paralelizar.

Comentarios finales

- El algoritmo de retropropagación es uno de los principales en machine-learning. Aun hoy en día se usa.
- Tedencias actuales: deep learning (resurgimiento de RNs)
- Diferencia modelo no-lineal vs modelo lineal.