**TALLER N° 3 – INTRODUCCIÓN A LAS REDES NEURONALES**

**LUIS CARLOS JORDÁN HURTADO**

**TRABAJO DE:**

**INTELIGENCIA ARTIFICIAL**

**PRESENTADO A:**

**CARLOS ALBERTO LONDOÑO LOAIZA**

**CARTAGO VALLE**

**CORPORACIÓN DE ESTUDIOS TECNOLÓGICOS DEL NORTE DEL VALLE**

**TECNOLOGIA EN SISTEMAS DE INFORMACIÓN**

**SEMESTRE V**

**2018**

# HISTORIA DE REDES NEURONALES ARTIFICIALES

Conseguir, diseñar y construir máquinas capaces de realizar procesos con cierta inteligencia ha sido uno de los principales objetivos de los científicos a lo largo de la historia. De los intentos realizados en este sentido se han llegado a definir las líneas fundamentales para la obtención de máquinas inteligentes: En un principio los esfuerzos estuvieron dirigidos a la obtención de autómatas, en el sentido de máquinas que realizaran, con más o menos éxito, alguna función típica de los seres humanos. Hoy en día se continúa estudiando en ésta misma línea, con resultados sorprendentes, existen maneras de realizar procesos similares a los inteligentes y que podemos encuadrar dentro de la llamada Inteligencia Artificial (IA). ()

## Mapa conceptual:

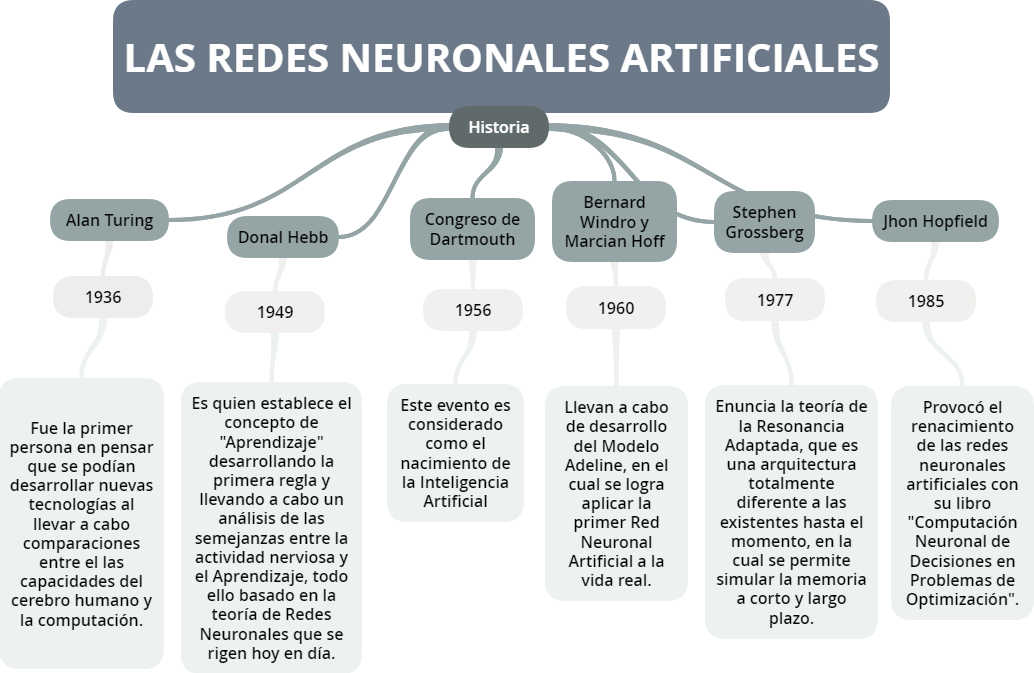


Ilustración 1 Mapa Mental Redes Neuronales Artificiales

# VENTAJAS de las rna

* **APRENDIZAJE:** Las RNA tienen la habilidad de aprender mediante una etapa que se llama etapa de aprendizaje. Esta consiste en proporcionar a la RNA datos como entrada a su vez que se le indica cuál es la salida (respuesta) esperada.
* **AUTO ORGANIZACIÓN:** Una RNA crea su propia representación de la información en su interior, descargando al usuario de esto, lo que en la vida real puede ser catalogado como interpretación.
* **TOLERANCIA A FALLOS:** Debido a que una RNA almacena la información de forma redundante, ésta puede seguir respondiendo de manera aceptable aun si se daña parcialmente.
* **FLEXIBILIDAD:** Una RNA puede manejar cambios no importantes en la información de entrada, como señales con ruido u otros cambios en la entrada (ej. si la información de entrada es la imagen de un objeto, la respuesta correspondiente no sufre cambios si la imagen cambia un poco su brillo o el objeto cambia ligeramente).
* **TIEMPO REAL:** La estructura de una RNA es paralela, por lo cual si esto es implementado con computadoras o en dispositivos electrónicos especiales, se pueden obtener respuestas en tiempo real.

La capacidad de aprendizaje adaptativo, es una de las características más atractivas de redes neuronales. Aprenden a llevar a cabo ciertas tareas mediante un entrenamiento con ejemplos ilustrativos. Como las redes neuronales pueden aprender a diferenciar patrones mediante ejemplos y entrenamientos, no es necesario elaborar modelos ni necesidad de especificar funciones de [distribución](http://www.monografias.com/trabajos11/travent/travent.shtml) de [probabilidad](http://www.monografias.com/trabajos11/tebas/tebas.shtml).

# DESVENTAJAS DE LAS RNA

• **Complejidad de aprendizaje** para grandes tareas, cuanto más cosas se necesiten que aprenda una red, más complicado será enseñarle.

• **Tiempo de aprendizaje elevado.** Esto depende de dos factores: primero si se incrementa la cantidad de patrones a identificar o clasificar y segundo si se requiere mayor flexibilidad o capacidad de adaptación de la red neuronal para reconocer patrones que sean sumamente parecidos, se deberá invertir más tiempo en lograr que la red converja a valores de pesos que representen lo que se quiera enseñar.

• **No permite interpretar lo que se ha aprendido**, la red por si sola proporciona una salida, un número, que no puede ser interpretado por ella misma, sino que se requiere de la intervención del programador y de la aplicación en si para encontrarle un significado a la salida proporcionada.

• **Elevada cantidad de datos para el entrenamiento**, cuanto más flexible se requiera que sea la red neuronal, más información tendrá que enseñarle para que realice de forma adecuada la identificación.

• **Padecen nuestra falta de hardware.** La capacidad de las redes neuronales radica en su habilidad de procesar información en paralelo (esto es, procesar múltiples pedazos de datos simultáneamente).

# aplicaciones de redes neuronales

## BIOLOGÍA

* Aprender más acerca del cerebro
* Obtención del modelo de la retina

## EMPRESA

* Evaluación de probabilidad de formaciones geológicas y petrolíferas
* Explotación de bases de datos
* Optimización de plazas y horarios en líneas de vuelo
* Reconocimiento de caracteres escritos

## MEDIO AMBIENTE

* Analizar tendencias y patrones
* Previsión del tiempo

## FINANZAS

* Previsión de la evolución de los precios
* Valoración del riesgo de los créditos
* Interpretación de firmas
* Identificación de falsificaciones

## MANUFACTURACIÓN

* Robots y sistemas de control (visión artificial y sensores de presión, temperatura, etc.)
* Control de producción en líneas de proceso

## MEDICINA

* Analizadores del habla para la ayuda de audición de sordos
* Monitorización en cirugía
* Predicción de reacciones adversas a los medicamentos
* Lectores de rayos X
* Diagnóstico y tratamiento a partir de síntomas y/o datos analíticos

## MILITARES

* Clasificación de las señales de radar
* Creación de armas inteligentes
* Reconocimiento y seguimiento en el tiro al blanco
* De la clasificación anterior podemos deducir que la mayoría de ellas consisten en realizar un reconocimiento de patrones: búsqueda, clasificación, reconstrucción.

# FUNCIÓN DE ACTIVACIÓN

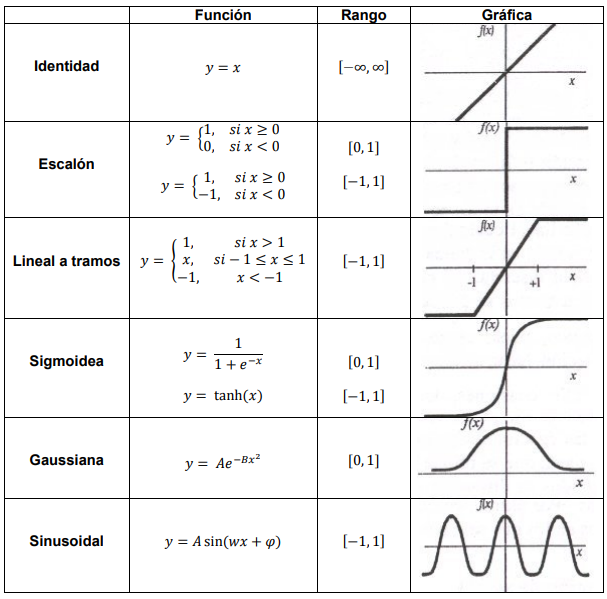
La función de activación determina el estado de activación actual de la neurona en base al potencial resultante 𝑛𝑒𝑡𝑖 y al estado de activación anterior de la neurona (𝑡 −1). El estado de activación de la neurona para un determinado instante de tiempo t puede ser expresado de la siguiente manera:

𝑎𝑖(𝑡) = 𝑓(𝑎𝑖 (𝑡 − 1), 𝑛𝑒𝑡𝑖 (𝑡))

Sin embargo, en la mayoría de los modelos se suele ignorar el estado anterior de la neurona, definiéndose el estado de activación en función del potencial resultante ℎ𝑖:

𝑎𝑖(𝑡) = 𝑓( 𝑛𝑒𝑡𝑖(𝑡))

Algunas de las funciones de activación más utilizadas en los distintos modelos de redes neuronales son: (Barrera, Cucei, n.d.)



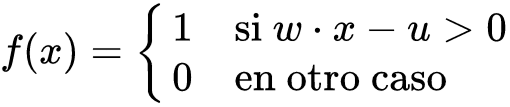
# PERCEPTRÓN

## HISTORIA

1957 – Frank Rosenblatt. Comenzó el desarrollo del Perceptrón. Esta es la red neuronal más antigua; utilizándose hoy en día para aplicación como reconocedor de patrones. Este modelo era capaz de generalizar, es decir, después de haber aprendido una serie de patrones podía reconocer otros similares, aunque no se le hubiesen presentado anteriormente. Sin embargo, tenía una serie de limitaciones, por ejemplo, su incapacidad para resolver el problema de la función OR-exclusiva y, en general, era incapaz de clasificar clases no separables linealmente. En 1959, escribió el libro Principios de Neurodinámica, en el que confirmó que, bajo ciertas condiciones, el aprendizaje del Perceptrón convergía hacia un estado finito (Teorema de Convergencia del Perceptrón). (Laredo, n.d., p. 2)

## FORMULA MATEMATICA

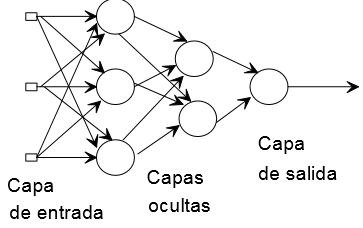
El perceptrón usa una matriz para representar las redes neuronales y es un discriminador terciario que traza su entrada **X** (un vector binario) a un único valor de salida **f(x)** (un solo valor binario) a través de dicha matriz.



Donde **W** es un vector de pesos reales y **W ⋅ X** es el producto escalar (que computa una suma ponderada). **U** es el ‘umbral’, el cual representa el grado de inhibición de la neurona, es un término constante que no depende del valor que tome la entrada.

El valor de **f(x)** (0 o 1) se usa para clasificar **X** como un caso positivo o un caso negativo, en el caso de un problema de clasificación binario. El umbral puede pensarse de como compensar la función de activación, o dando un nivel bajo de actividad a la neurona del rendimiento. La suma ponderada de las entradas debe producir un valor mayor que **U** para cambiar la neurona de estado 0 a 1. (Sinfallas, 2017)

## ESTRUCTURA

Perceptrón multicapa Este modelo se compone de la siguiente manera: o Capa de entrada: sólo se encarga de recibir las señales de entrada y propagarla a la siguiente capa.

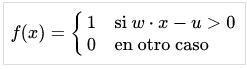
Capa de salida: proporciona al exterior la respuesta de la red para cada patrón de entrada. Capas ocultas: realizan un procesamiento no lineal de los datos de entrada.

## ¿PÁRA QUÉ SE USA?

De lo cual se desarrolla un algoritmo capaz de generar un criterio para seleccionar un sub-grupo a partir de un grupo de componentes más grande. La limitación de este algoritmo es que si dibujamos en un Plot estos elementos, se deben poder separar con un Hiperplano únicamente los elementos “deseados” discriminándolos (separándolos) de los “no deseados”.

## FUNCIÓN DE ACTIVACIÓN

El perceptrón usa una matriz para representar las redes neuronales y es un discriminador terciario que traza su entrada **X** (un vector binario) a un único valor de salida **f(x)** (un solo valor binario) a través de dicha matriz.



Donde **W** es un vector de pesos reales y **W** ⋅ **X** es el producto escalar (que computa una suma ponderada). **U** es el ‘umbral’, el cual representa el grado de inhibición de la neurona, es un término constante que no depende del valor que tome la entrada.

El valor de **f(x)** (0 o 1) se usa para clasificar **X** como un caso positivo o un caso negativo, en el caso de un problema de clasificación binario. El umbral puede pensarse de como compensar la función de activación, o dando un nivel bajo de actividad a la neurona del rendimiento. La suma ponderada de las entradas debe producir un valor mayor que **U** para cambiar la neurona de estado 0 a 1.

## ENTRENAMIENTO DE UN PERCEPTRÓN

En el perceptrón, existen dos tipos de aprendizaje, el primero utiliza una tasa de aprendizaje mientras que el segundo no la utiliza. Esta tasa de aprendizaje amortigua el cambio de los valores de los pesos.

Los dos tipos de aprendizaje difieren en este paso. Para el primer tipo de aprendizaje, utilizando tasa de aprendizaje, utilizaremos la siguiente regla de actualización de los pesos:

**W(J)′ = W(J) + Α(Δ − Y) X(J)**

Para el segundo tipo de aprendizaje, sin utilizar tasa de aprendizaje, la regla de actualización de los pesos será la siguiente:

**W(J)′ = W(J) + (Δ − Y) X(J)**

Por lo cual, el aprendizaje es modelado como la actualización del vector de peso después de cada iteración, lo cual sólo tendrá lugar si la salida **Y** difiere de la salida deseada **δ**. Para considerar una neurona al interactuar en múltiples iteraciones debemos definir algunas variables más:

**X*i*** denota el vector de entrada para la iteración *i*

**W*i*** denota el vector de peso para la iteración *i*

**Y*i*** denota la salida para la iteración *i*

**Dm = {(x1, y1), … ,(xm, ym)}** denota un periodo de aprendizaje de **M** iteraciones.

En cada iteración el vector de peso es actualizado como sigue:

Para cada pareja ordenada **(X, Y)** en **Dm = {(X*1*, Y*1*), … ,(X*m*, Y*m*)}**

Pasar (**X*i*, Y*i*, W*i***) a la regla de actualización **w(j)′ = w(j) + α(δ − Y) x(j)**

El periodo de aprendizaje **D*m*** se dice que es separable linealmente si existe un valor positivo **Y** y un vector de peso **W** tal que: **Y*i* ⋅ (⟨W, X*i*⟩ + U) > Y** para todos los ***i***.

Novikoff probó que el algoritmo de aprendizaje converge después de un número finito de iteraciones si los datos son separables linealmente y el número de errores está limita

Sin embargo si los datos no son separables linealmente, la línea de algoritmo anterior no se garantiza que converja.

## ALGORITMO

El algoritmo de aprendizaje es el mismo para todas las neuronas, todo lo que sigue se aplica a una sola neurona en el aislamiento. Se definen algunas variables primero:

El **X(J)** denota el elemento en la posición **J** en el vector de la entrada.

El **W(J)** el elemento en la posición **J** en el vector de peso.

El **Y** denota la salida de la neurona.

El **δ** denota la salida esperada.

El **α** es una constante tal que **0 < α < 1**

## Evidencias

1. modelo para clasificar hombres y mujeres según su peso y su estatura\*. Los siguientes datos obtuve de amigos míos:

Estatura. Peso. Género.

170 56 Mujer(1)

172 63 Hombre(0)

160 50 Mujer(1)

170 63 Hombre(0)

174 66 Hombre(0)

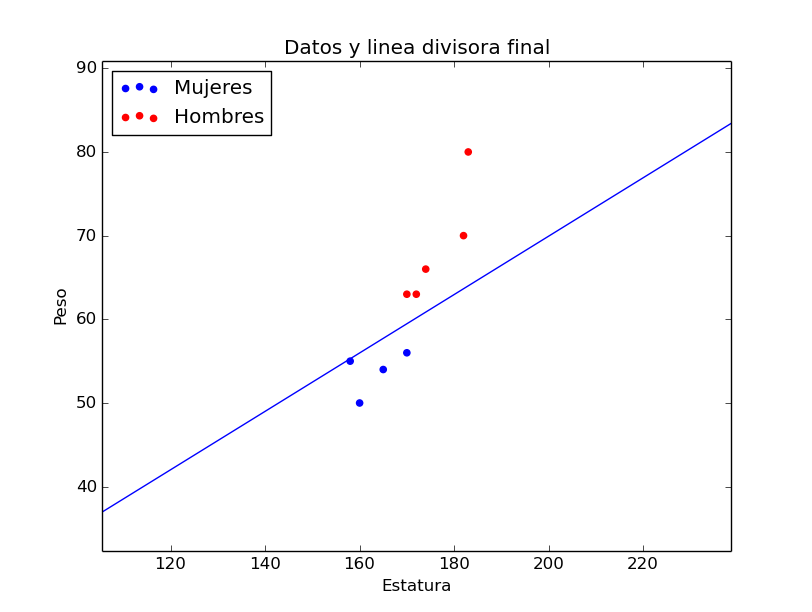
158 55 Mujer(1)

183 80 Hombre(0)

182 70 Hombre(0)

165 54 Mujer(1)

Resultado



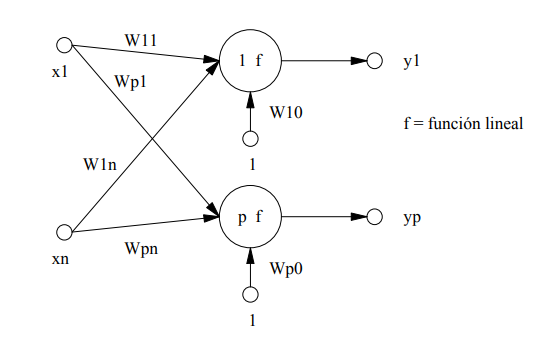
(Estrada, 2016)

# ADALINE

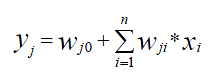
## HISTORIA

1960 – Bernard Windrow y Marcial Hoff. Desarrollaron el modelo Adaline (ADAptative LINear Elements). Esta fue la primera red neuronal aplicada a un problema real (filtros adaptativos para eliminar ecos en las líneas telefónicas) que se ha utilizado comercialmente durante varias décadas.

## Fórmula matemática



Aplicada a estructuras Lineales



## ESTRUCTURA

Red Adaline está formada por un elemento denominado Combinador Adaptativo Lineal (ALC). La salida lineal obtenida del ALC es aplicada a un Conmutador Bipolar.

El Umbral se representa a través de una conexión ficticia de peso W0 (b). La red Adaline puede tomar valores continuos.

## ¡PARA QUÉ SE USA?

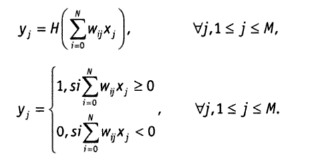
La principal aplicación de las redes tipo Adaline se encuentra en el campo de procesamiento de señales. Concretamente en el diseño de filtros capaces de eliminar ruido en señales portadoras de información.

Otra aplicación es la de los filtros adaptativos: Predecir el valor futuro de una señal a partir de su valor actual.



## Función de activación

La función de activación de las neuronas de salida es de tipo Escalón.



## Entrenamiento de una red Adaline

El entrenamiento de la red consiste en la aplicación consecutiva de la regla de aprendizaje para un patrón completo de entrenamiento

El entrenamiento comprende la aplicación de la regla de Widrow-Hoff (determinación de las diferencias en los pesos), actualización de los pesos y determinación del error.

Este proceso se realiza por medio de la aplicación de la entrada a la red, para un patrón completo de entrenamiento, o un vector de entrada

Por medio de Matlab, se puede realizar el entrenamiento por medio de learnwh, realizando las funciones siguientes:

**A = simulin (P,W,b)**

**E = T - A**

**[dW,db] = learnwh (P,E,lr)**

**W = W + dW**

**b = b + dW**

**lr** es la tasa de aprendizaje. Si es grande, el aprendizaje es rápido, pero si es demasiado grande, el aprendizaje es inestable y puede incrementarse el error.

**lr = maxlinlr( P ); % si no se utiliza bias**

**lr = maxlinlr( P, ‘bias’ ); %si se utiliza bias**

**W = W + dW;**

**b = b + db;**

Sin embargo, si se procede por medio de las funciones anteriores, es necesario realizar un número de repeticiones suficiente para hacer que el error calculado sea igual a cero a través de la aplicación de la regla W-H para un patrón completo de entrenamiento.

En Matlab, se puede realizar el proceso de entrenamiento de una capa de red lineal adaline por medio de la función

trainwh():

Esta función aplica repetidamente entradas a la red lineal, calcula el error en base al vector objetivo, y asigna pesos y umbrales

La sintaxis de la función es:

tp = [disp\_frec max\_epoch err\_goal lr ]

[W,b,ep,tr] = trainwh(W,b,P,T,tp)

**tp** es un parámetro que indica la frecuencia de progreso en el entrenamiento, el máximo de repeticiones, el error mínimo permisible y la tasa de aprendizaje (ver Aprendizaje) y puede contener valores nulos o NAN.

La función regresa los pesos W, los umbrales b, el número de repeticiones usado ep y un registro del error de entrenamiento tr.

## Algoritmo

* **Cálculo de Pesos Óptimos**

Modificación de Pesos para tratar de reducir la diferencia entre la salida deseada y la actual (para cada patrón). Se denomina LMS: minimiza el Error Cuadrático Medio sobre todos los patrones de entrenamiento.

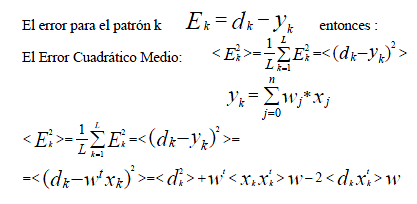
Sea el conjunto de entrenamiento: (X,D): patrones de entrada y salidas deseadas.

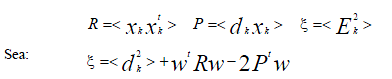
X ---- conjunto de L vectores de dimensión n.

D ---- conjunto de L vectores de dimensión m (en este caso m=1). Salida Deseada.

Y ---- conjunto de L vectores de dimensión m (en este caso m=1). Salida Obtenida

Se trata de minimizar: Sea Yk la salida obtenida para el patrón k.





# REDES DE RETROPROPAGACIÓN (BACKPROPAGATION)

Al hablar de redes de retropropagación o redes de propagación hacia atrás hacemos referencia a un algoritmo de aprendizaje más que a una arquitectura determinada. La retropropagación consiste en propagar el error hacia atrás, es decir, de la capa de salida hacia la capa de entrada, pasando por las capas ocultas intermedias y ajustando los pesos de las conexiones con el fin de reducir dicho error. (Magomar, n.d.)

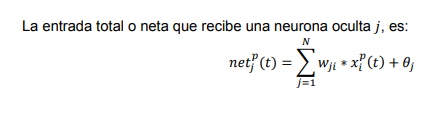
Las redes multicapa son capaces de calcular una gama más amplia de funciones que las redes compuestas por una única capa de elementos. Pero el esfuerzo computacional necesario para encontrar la combinación correcta de pesos se incrementa sustancialmente cuando se consideran más parámetros y topologías más complejas. Se ha visto en la sección anterior que el aprendizaje de un perceptron podía interpretarse gráficamente como la búsqueda de un mínimo de una función de error en el espacio de los pesos. Cada combinación de pesos produce una función lógica que difiere de la función que se desea aprender en un número de puntos. El aprendizaje se define como la búsqueda de la combinación que minimiza ese número. Se vio también que la función de error de un perceptrón aislado era una función simple con un único mínimo global. Pero el algoritmo de aprendizaje de una red neuronal sin realimentación es en su caso general mucho más difícil de definir, ya que la optimización de una función de error discontinua no es un problema sencillo. Sin embargo existen procedimientos numéricos eficaces para buscar mínimos en funciones derivables. Por esta razón, se relaja cl problema definiendo un perceptron continuo, en cl que la función de activación no sea la función signo, sino una función continua y derivable parecida a la función signo. Aun así, aparecen mínimos locales en las superficies, lo que complica también el proceso.

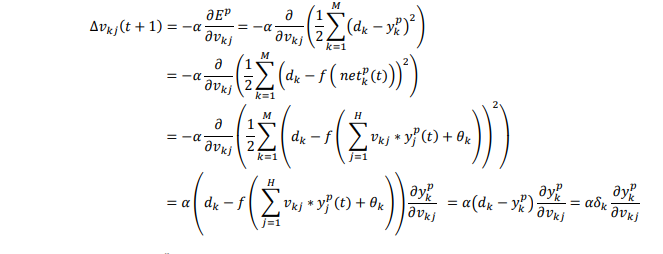
## ¿Para qué se usa?

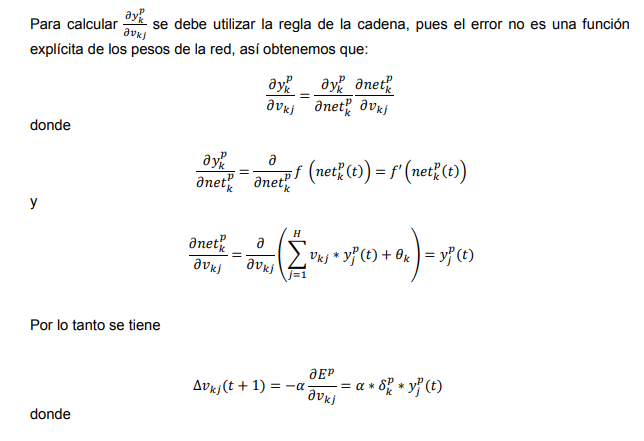
Las redes multicapa son capaces de calcular una gama más amplia de funciones que las redes compuestas por una única capa de elementos. Pero el esfuerzo computacional necesario para encontrar la combinación correcta de pesos se incrementa sustancialmente cuando se consideran más parámetros y topologías más complejas. Se ha visto en la sección anterior que el aprendizaje de un perceptron podía interpretarse gráficamente como la búsqueda de un mínimo de una función de error en el espacio de los pesos. Cada combinación de pesos produce una función lógica que difiere de la función que se desea aprender en un número de puntos. El aprendizaje se define como la búsqueda de la combinación que minimiza ese número. Se vio también que la función de error de un perceptrón aislado era una función simple con un único mínimo global. Pero el algoritmo de aprendizaje de una red neuronal sin realimentación es en su caso general mucho más difícil de definir, ya que la optimización de una función de error discontinua no es un problema sencillo. Sin embargo existen procedimientos numéricos eficaces para buscar mínimos en funciones derivables. Por esta razón, se relaja cl problema definiendo un perceptron continuo, en cl que la función de activación no sea la función signo, sino una función continua y derivable parecida a la función signo. Aun así, aparecen mínimos locales en las superficies, lo que complica también el proceso.

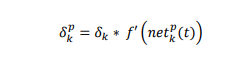
# algoritmo backpropagation

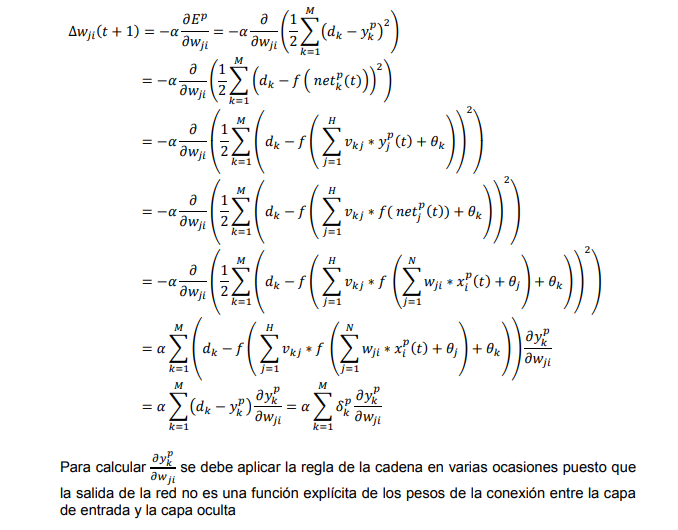
Paso 1. Inicializar los pesos de la red con valores pequeños aleatorios. Paso 2. Presentar un patrón de entrada y especificar la salida deseada que debe generar la red. Paso 3. Calcular la salida actual de la red. Para ello se presentan las entradas a la red y se calcula la salida de cada capa hasta llegar a la capa de salida, ésta será la salida de la red. Los pasos son los siguientes: Se calculan las entradas netas para las neuronas ocultas procedentes de las neuronas de entrada. Para una neurona j oculta: ∑= = + N i h pi j h ji h pj net w x 1 θ 7 en donde el índice h se refiere a magnitudes de la capa oculta; el subíndice p, al p-ésimo vector de entrenamiento, y j a la j-ésima neurona oculta. El término θ puede ser opcional, pues actúa como una entrada más. Se calculan las salidas de las neuronas ocultas: ( ) h pj h y pj = f j net . Se realizan los mismos cálculos para obtener las salidas de las neuronas de salida: ∑= = + L j pj k o kj o pk net w y 1 0 θ ( ) 0 o pk k pk y = f net Paso 4 . Calcular los términos de error para todas las neuronas. Si la neurona k es una neurona de la capa de salida, el valor de la delta es: ( ) ( ) 0 ´ o pk o pk pk pk k δ = d − y f net La función f debe ser derivable. En general disponemos de dos formas de función de salida: La función lineal: k jk jk f (net ) = net La función sigmoidal: jk net k jk e f net − + = 1 1 ( ) La selección de la función depende de la forma que se decida representar la salida: si se desea que las neuronas de salida sean binarias, se utiliza la función sigmoidal, en otros casos, la lineal. Para una función lineal, tenemos: ´=1 o k f , mientras que la derivada de una función sigmoidal es: ´ (1 ) (1 ) pk pk o k o k o k f = f − f = y − y por lo que los términos de error para las neuronas de salida quedan: ( ) pk pk o pk δ = d − y para la salida lineal. ( ) (1 ) pk pk pk pk o pk δ = d − y y − y para la salida sigmoidal. Si la neurona j no es de salida, entonces la derivada parcial del error no puede ser evaluada directamente, por tanto se obtiene el desarrollo a partir de valores que son conocidos y otros que pueden ser evaluados. La expresión obtenida en este caso es: = ∑ k pk kj h pj h j h pj f net w0 0 δ ´( ) δ donde observamos que el error en las capas ocultas depende de todos los términos de error de la capa de salida. De aquí surge el término propagación hacia atrás. Paso 5. Actualización de los pesos: para ello utilizamos un algoritmo recursivo, comenzando por las neuronas de salida y trabajando hacia atrás hasta llegar a la capa de entrada, ajustando los pesos de la siguiente forma: 8 Para los pesos de las neuronas de la capa de salida: ( 1) ( ) ( 1) 0 w t + = w t + ∆w t + o kj o kj kj pj o pk o kj ∆w (t +1) =αδ y Para los pesos de las neuronas de la capa oculta: w (t +1) = w (t) + ∆w (t +1) h ji h ji h ji pi h pj h ji ∆w (t +1) = αδ x En ambos casos, para acelerar el proceso de aprendizaje se puede añadir un término momento. Paso 6. El proceso se repite hasta que el término de error ∑= = M k Ep pk 1 2 2 1 δ resulta aceptablemente pequeño para cada uno de los patrones aprendidos [4][5].











# TEOREMA DE KOLMOGOROV

La ley cero-uno de Kolmogórov es un teorema de la teoría de las probabilidades. Llamado así en honor al matemático ruso Andréi Kolmogórov, dice que la probabilidad de cierto tipo de eventos llamados eventos de cola (tail event en inglés) es cero o uno. Los eventos de cola son aquellos eventos definidos por una sucesión infinita de eventos independientes, pero que son independientes de cualquier subconjunto finito de estos.

Por ejemplo, supongamos infinitas tiradas de una moneda. El evento: "que salga en total una cantidad finita de caras" es independiente de cualquier número finito de tiradas. Examinando una cantidad finita de tiradas no podemos concluir nada respecto a si la cantidad de caras fue finita o infinita.

La definición que Kolmogorov establece en 1933 resuelve el problema desde un punto de vista formal y práctico y es, sin duda alguna, lo que ha configurado el gran desarrollo de la teoría de la probabilidad a partir de ese momento. El problema que nos concierne es cómo hacer llegar este bello concepto, altamente abstracto en su expresión formal, a personas de otras disciplinas que no tienen una base matemática para entenderlo.

Para ello basta recordar que muchos modelos matemáticos parten de un simple hecho de la vida que deseamos explicar. Y muchas veces sucede que se desarrolla toda una teoría a partir de ese modelo abstracto, y luego resulta aplicable a muchas y muy variadas situaciones. Este es el caso de la probabilidad.

# Bibliografía

Agarwal, M. (14 de 12 de 2017). *becominghuman*. Obtenido de becominghuman: https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199

Barrera, J. A.-T. (2009). *CUCEI.* Obtenido de http://www.cucei.udg.mx/sites/default/files/pdf/toral\_barrera\_jamie\_areli.pdf

Barrera, J. A.-T. (s.f.). *Cucei*. Obtenido de http://www.cucei.udg.mx/sites/default/files/pdf/toral\_barrera\_jamie\_areli.pdf

BiBling. (s.f.). *BiBling*. Obtenido de http://bibing.us.es/proyectos/abreproy/3934/fichero/Documentos+de+inter%C3%A9s%252FAprendizaje.pdf

Estrada, P. (4 de 3 de 2016). *pabloem.* Obtenido de http://pabloem.github.io/Programando-una-red-neuronal/

Galván, I. y. (s.f.). Obtenido de http://ocw.uc3m.es/ingenieria-informatica/redes-de-neuronas-artificiales/transparencias/material-de-clase.-tema-2

Laredo, I. N. (s.f.). *IT Nuevo Laredo*. Obtenido de IT Nuevo Laredo: http://www.itnuevolaredo.edu.mx/takeyas/apuntes/Inteligencia%20Artificial/Apuntes/tareas\_alumnos/RNA/Redes%20Neuronales2.pdf

*Magomar*. (s.f.). Obtenido de http://magomar.webs.upv.es/rna/tutorial/RNA\_backprop.html

Medina, E., Cubides, H., & Salazar, J. y. (03 de 06 de 2015). Obtenido de http://es.scribd.com/doc/52002261/Adeline-1

Panedo, M. F. (s.f.). *Sistemas Conexionales.*

Raquel Flórez López, J. M. (2008). *Google Books.* Obtenido de Google Books: https://books.google.com.co/books?id=X0uLwi1Ap4QC&dq=funcion+de+activacion+adaline&source=gbs\_navlinks\_s

Rodríguez Herrador, L. (s.f.). Obtenido de http://bibing.us.es/proyectos/abreproy/3934/fichero/Documentos+de+inter%C3%A9s%252FAprendizaje.pdf

Saavedra, N. J. (1987). *LA AXIOMÁTICA DE KOLMOGOROV: FUNDAMENTO DE LA TEORÍA DE LA PROBABILIDAD.* Rev. colomb. estad., Volumen 8, Número 15-16, 1987. eISSN 2389-8976. Imprimir ISSN 0120-1751. .

*Sinfallas*. (14 de noviembre de 2017). Obtenido de https://sinfallas.wordpress.com/2017/11/14/el-perceptron-y-perceptron-multicapa-que-es-y-con-que-se-come/

Solano, R. C. (11 de 2010). *Modelos de Redes Neuronales*. Obtenido de http://ronaldchavezblog.blogspot.com.co/2010/11/red-adeline.html

Yáñez Márquez, C. S. (2007). *repositoriodigital*. Obtenido de http://www.repositoriodigital.ipn.mx/handle/123456789/8628