**2- VENTAJAS DE LAS REDES NEURONALES**

**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, quitándole esta tarea al usuario.

**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 (por ejemplo 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.

**Fácil inserción dentro de la tecnología existente**. Debido a que una red puede ser rápidamente entrenada, comprobada, verificada y trasladada a una implementación hardware de bajo costo, es fácil insertar RNA para aplicaciones específicas dentro de sistemas existentes (chips, por ejemplo). De esta manera, las redes neuronales se pueden utilizar para mejorar sistemas de forma incremental, y cada paso puede ser evaluado antes de acometer un desarrollo más amplio.

**DESVENTAJAS DE LAS REDES NEURONALES**

**Tiempo De Convergencia**. Las aplicaciones reales pueden llegar a tener miles de ejemplos en el conjunto de entrenamiento y ello requiere días de tiempo de cálculo. Además la backpropagation es susceptible de fallar en el entrenamiento, es decir, la red puede que nunca llegue a converger.

**Complejidad de aprendizaje para grandes tareas**, cuanto más cosas se necesiten que aprenda una red, mas 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 mas 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 mas flexible se requiera que sea la red neuronal, mas información tendrá que enseñarle para que realice de forma adecuada la identificación.

**Otros** problemas con las redes neuronales son la falta de reglas definitorias que ayuden a realizar una red para un problema dado.

**3- APLICACIONES**

**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

**4- Función de activación (activation function).**

que tiene un “estado de activación”. Las neuronas artificiales también tienen diferentes estados de activación; algunas de ellas solamente dos, al igual que las biológicas, pero otras pueden tomar cualquier valor dentro de un conjunto determinado.

La función activación calcula el estado de actividad de una neurona; transformando la entrada global (menos el umbral, Θi) en un valor (estado) de activación, cuyo rango normalmente va de (0 a 1) o de (–1 a 1). Esto es así, porque una neurona puede estar totalmente inactiva (0 o –1) o activa (1). La función activación, es una función de la entrada global (gini) menos el umbral (Θi).

Función escalón : se utiliza cuando las salidas de la red son binarias. La salida de una neurona se activa sólo cuando el estado de activación es mayor o igual que cierto valor umbral t que representa la mínima entrada total ponderada necesaria para provocar la activación de la neurona.

La función de activación combina el potencial postsinaptico, que nos proporciona la función de propagación, con el estado actual de la neurona para conseguir el estado futuro de activación de la neurona. Sin embargo, es muy común que las redes neuronales no tomen su propio estado como un parámetro y que por tanto no se considere . Esta función es normalmente creciente monótona y podemos citar las funciones más comunes:

**Lineal:**

Algunas redes neuronales usan esta función de activación como el Adeline por su eficiencia y facilidad.

**Escalón:**

Esta función es la más usada para redes neuronales binarias ya que no es lineal y es muy simple. Algunas redes que usan esta función son el Perceptrón y Hopfield. Para redes que trabajan en el rango [-1,1] se usa la funcion signo

**Hiperbólicas o tangenciales:**

Las redes con salidas continuas, como el Perceptron multicapa con retropropagación, usan esta función ya que su algoritmo de aprendizaje necesita una función derivable

**5- PERCEPTRON**

La primera red neuronal conocida, fue desarrollada en 1943 por Warren McCulloch y Walter Pitts -Suma de las señales de entrada, multiplicadas por unos valores de pesos escogidos aleatoriamente, la entrada es comparada con un patrón preestablecido para determinar la salida de la red, Si en la comparación, la suma de las entradas multiplicadas por los pesos es mayor o igual que el patrón preestablecido la salida de la red es uno (1), en caso contrario la salida es cero (0).

La red tipo Perceptrón fue inventada por el sicólogo Frank Rosenblatt en el año 1957. El primer modelo de Perceptrón fue desarrollado en un ambiente biológico imitando el funcionamiento del ojo humano. El fotoperceptrón: era un dispositivo que respondía a señales ópticas. El Perceptron es la más antigua red neuronal, y se usa hoy en día de varias formas para la aplicación como reconocedor de patrones.

Para utilizar el perceptron se usa la siguiente formula



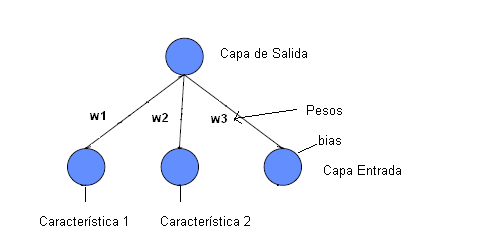
Este net es la suma de las multiplicaciones del valor de salida de los nodos de entrada por su correspondiente peso. Por ejemplo, si tuviéramos un patrón de dos características más el valor de salida del bias que es 1: p1 = [2,3,1] y tuviéramos un vector de pesos w=[0.2,0.3,0.4], el net sería

net = (2\*0.2)+(3\*0.3)+(1\*0.4)

net = 0.4 + 0.9 + 0.4

net = 1.7

**Estructura del Perceptron**

****

Como se puede apreciar, el perceptrón está formado por dos capas, una de entrada con un número de nodos determinado y una de salida con un sólo nodo el cuál se encuentra conectado a cada uno de los nodos de la capa de entrada mediante una conexión que está valuada con un peso (w1, w2 y w3). Existe un nodo extra llamado bias el cuál no tiene contacto con el exterior y su valor siempre es 1. Cabe hacer la aclaración que algunos autores no toman en cuenta la capa de entrada debido a que en ésta no se lleva cabo ningún procesamiento de la información, simplemente sirve como enlace con el exterior de la red neuronal y su única tarea es recibir los valores de entrada del exterior y pasárselos al nodo de la capa de salida. Para estos autores el perceptrón consta de una capa. Para este artículo, el perceptrón consta de dos capas: una de entrada y una de salida.

Para la primera capa, tendremos igual número de nodos que número de características en los patrones a analizar más el nodo del bias, o bien, dicho de otro modo, tendremos igual número de nodos que número de elementos en nuestro vector que representa al patrón más el nodo del bias. Es decir, si tenemos un patrón representado por el siguiente vector p1 = [a1,a2], entonces tendremos en nuestra primera capa dos nodos de entrada más el nodo del bias cuya salida siempre será 1. Los dos nodos de la capa de entrada relacionados con las características del patrón, deberán ser alimentados con los valores respectivos de los patrones que se estén usando.

La función que representa el comportamiento ideal de una neurona es la función escalón: dependiendo de la entrada se activa la conexión entre neuronas (salida = 1) o no (salida = 0). Esta decisión puede depender de un cierto umbral (que la salida sea capaz de superarlo), representado en las expresiones anteriores por que introducir este término negativo equivale a considerar que la función vale uno a partir de q y no a partir de cero como se representa en la tabla.

Aunque éste sea el comportamiento ideal, en la práctica se suele utilizar la función sigmoidea. Esto se debe a que es una función muy parecida al escalón y derivable en el entorno de cualquier punto del eje x. Como se verá, esto es necesario en el algoritmo de retropropagación ya que está basado en la minimización del error con respecto al valor de los pesos, por lo que hay que derivar e igualar a cero.

El algoritmo de aprendizaje es de tipo supervisado. En el proceso de entrenamiento, el Perceptron se expone a un conjunto de patrones de entrada, y los pesos de la red son ajustados de forma que al final del entrenamiento se obtenga las salidas esperadas para cada uno de esos patrones de entrada.

Algoritmo del perceptron DISCRETO

VARIABLES

I (nº de iteraciones)

K (contador de ejemplos)

W[i:1..n+1] (pesos del perceptron)

X[i:1..n] (valores de la entrada en el ejemplo actual)

Y (valor de la salida en el ejemplo actual)

SP (salida del perceptrón con la entrada del ejemplo actual)

Error (error en el ejemplo actual)

T (Variable para indicar si el vector es correcto)

PASO 1: Inicialización de variables

PASO 2: Bucle de iteraciones (hasta condición de parada)

Paso 2.1: Bucle de paso por todos los ejemplos

2.1.0 Leer valores del ejemplo

2.1.1 Calcular error en ese ejemplo

2.1.2 Actualizar los pesos según el error de ese ejemplo

2.1.2.1 Actualizar los pesos de las entradas

2.1.2.2 Actualizar el bias (= -umbral)

2.1.3 Incrementar contador de ejemplos

Paso 2.2.: Ver si el vector de pesos es correcto

Paso 2.3.: Incrementar el contador de iteraciones

PASO 3: Salida

PASO 1: Inicialización de variables

I=0;

Para i=1..n+1

W[i]=<nº aleatorio (normalmente entre –1 y 1)> (muy importante que el nº sea aleatorio,

distinto en cada ejecución)

T=false

PASO 2: Bucle de iteraciones

Mientras (I < NºMaxIteraciones) y (no T)

Paso 2.1: Bucle de paso por todos los ejemplos

Abrir(<fichero de ejemplos>)

K=0

Mientras no este en el final del fichero

2.1.0 Leer valores del ejemplo

Leer(x[i] (i=1..n), y)

2.1.1 Calcular error en ese ejemplo

SP = W[n+1] + ∑ x[i] \* W[i] (salida del perceptrón con la entrada del ejemplo h)

Si SP < 0, entonces SP= 0, en otro caso SP=1

Error = y - SP (diferencia entre el valor de y en el ejemplo h y SP)

2.1.2 Actualizar los pesos según el error de ese ejemplo

2.1.2.1 Actualizar los pesos de las entradas

Para cada i=1..n

W[i] = W[i] + x[i]\*Error

2.1.2.2 Actualizar el bias (= -umbral)

W[n+1]=W[n+1] + Error

2.1.3 Incrementar contador de ejemplos

k=k+1

Cerrar(<fichero de ejemplos>)

Paso 2.2.: Ver si el vector de pesos es correcto

T= True

Abrir(<fichero de ejemplos>)

Mientras ( no este en el final del fichero y T)

Leer(x[i] (i=1..n), y)

SP = W[n+1] + ∑ x[i] \* W[i]

Si SP < 0, entonces SP= 0, en otro caso SP=1

Error = y – SP

Si Error ≠ 0, T = False

Cerrar(<fichero de ejemplos>)

Paso 2.3.: Incrementar el contador de iteraciones

I=I+1

PASO 3: Salida

Escribe( “El perceptrón aprendido es el de pesos:” )

Para i=1..n

Escribe(“ W”,i,”=”,W[i])

Escribe(“Con bias =”, W[n+1])

El perceptrón multicapa es la red neuronal mas utilizada actualmente; sus principales ventajas son el procesamiento en paralelo, la tolerancia al ruido, el aprendizaje mediante ejemplos y la generalización. Para implementar el perceptrón multicapa no se necesita construir algoritmos complejos, como en los sistemas secuenciales, solamente se debe entrenar la red, mostrándole un grupo de ejemplos de entrada con su salida.

El perceptrón multicapa puede ser utilizado en diversas aplicaciones en donde se necesite procesar información compleja, entre estas se puede mencionar codificación de datos, reconocimiento del habla, reconocimiento de imágenes visuales, control de robots, reconocimiento de caracteres, pronóstico de eventos y control automático de procesos.

Por ejemplo en la codificación de datos puede utilizarse un perceptrón multicapa con un número de entradas igual o menor al número de salidas. Para codificar información digital debe utilizarse una tabla con los diferentes valores de entrada con sus respectivos códigos de salida; los datos de la tabla deben utilizarse para el entrenamiento de la red, así cuando a la red se le presente una entrada, se obtendrá el correspondiente código salida.

Una aplicación muy común del perceptrón multicapa es el reconocimiento de imágenes visuales. En esta aplicación, los píxeles que forman la imagen son usados como entradas a la red neuronal y los valores de salida deben corresponder con la clase a la que pertenece esta imagen. El entrenamiento debe realizarse con varios ejemplos de cada clase; estos ejemplos deben seleccionarse adecuadamente, para que la red pueda obtener las características que diferencian una clase de otra. Para mejorar la eficiencia de la red, generalmente se aplica algún tratamiento a la imagen antes de ingresarse a la red neuronal.

Otra aplicación del perceptrón multicapa es el control automático de un proceso industrial; en este caso, las salidas de los sensores son conectadas a la entrada de la red neuronal y las salidas de la red son conectadas a los actuadores. La tabla usada en el entrenamiento de la red debe contener todos los eventos posibles en el proceso, con su respectiva acción, de esta forma la red responderá con la acción adecuada a cada evento del proceso. La ventaja de usar redes neuronales en esta aplicación es que en caso de falla de algún sensor o ruido en la entrada, la red responderá con la acción adecuada, ya que la salida será la correspondiente al ejemplo que tenga la mayor similitud con esta entrada.

En muchas aplicaciones el proceso de entrenamiento podría ser tardado, según la complejidad de la información a clasificar, pero este entrenamiento es realizado solamente una vez; después de que la red haya sido entrenada, esta responde de forma rápida a las entradas que se le presentan.

El perceptrón multicapa puede ser utilizado en un gran número de aplicaciones, éste proporciona una solución aceptable en relativamente poco tiempo de desarrollo.

**6- ADALINE**

En 1959, Bernard Widrow y Marcian Hoff de la universidad de Stanford desarrollaron un modelo llamado “ADALINE” y “MADALINE”. ADALINE proviene de ADAptive LINear Element y MADALINE de Multiple ADAptive LINear Element (Many Adalines).

ADALINE fue desarrollado para el reconocimiento de patrones binarios, por ejemplo predecir el siguiente bit en una línea telefónica.

ADALINE funciona tomando la suma de los pesos de las entradas y produce una salida con 0 o 1 dependiendo si pasa o no un umbral, esto haciendo analogía al funcionamiento de una neurona que se dispara si la actividad total procedente de las conexiones con las otras neuronas sobrepasa un nivel

Varias ADALINE pueden ser organizadas en capas de tal manera que se obtengan grandes arquitecturas formando una red MADALINE la cual produce funciones más complicadas.

MADALINE fue la primera red neuronal aplicada a un problema real, se uso como un filtro para eliminar el eco en las líneas telefónicas.

Formula matemática

Generalmente se compone de una sola capa de n neuronas ( por tanto n valores de salida ) con m entradas con las siguientes características:

Las m entradas representan un vector x de entrada que pertenece al espacio {\ R^m

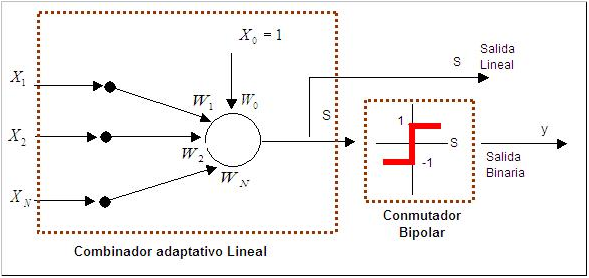
Por cada neurona, existe un vector w de pesos sinápticos que indican la fuerza de conexión entre los valores de entrada y la neurona. En la práctica representan la ponderación de cada entrada sobre la neurona. Una constante **θ.**

La salida y de la neurona se representa por la función de activación, que se define como:

{\displaystyle y=\sum \_{i=1}^{n}x\_{i}w\_{i}+\theta }



**ESTRUCTURA ADALINE**

****

El entrenamiento del ADALINE está basado en la regla LMS (Least Mean Square) que busca minimizar el error cuadrático medio por medio de la regla delta.

Dado un vector de entrada (x0, ..., xn) donde x0 =1, el correspondiente valor de pesos (w0, ..., wn) y el valor deseado de salida d, el error cuadrático es el siguiente:



El objetivo del método es minimizar dicho error mediante la modificación del vector de pesos (w0, ..., wn) sumándole un **w** de tal forma que nos acerquemos al error mínimo en la dirección del gradiente negativo, es decir, lo más rápidamente posible.

El procedimiento de derivación se presenta a continuación:



Por lo que el error local será reducido más rápidamente si se ajustan los pesos de acuerdo a la regla delta:



**Usos de Res Adaline**

La Red Adaline es un mecanismo físico capaz de realizar aprendizaje es un elemento Combinador Adaptativo, recibe un conjunto de entradas y las combina para producir una salida entonces ADALINE es un modelo de aprendizaje.

Las redes Adaline se basan en el objetivo de evaluar  de la forma más  correcta la salida, para de esta manera poder minimizar el error medio cuadrático, por tanto son muy similares al Perceptrón, lo único en lo que cambian es la transferencia en la salida.

Adaline tiene una considerable diferencia con respecto a Perceptron, puesto que en la modificación de los pesos que se dan en la preparación, la Red Adaline tiene muy presente el valor de corrección de la salida estimada con relación a la esperada

**El algoritmo de entrenamiento** se presenta a continuación:

1.Inicializar pesos (w1, ..., wn) y threshold (w0)

2.Presentar vector de entrada (x1,...,xn) y la salida deseada d(t)

3.Calcular la salida



donde Fh(a) = 1 sí a>0 y

= -1 sí a<=0

4. Adaptar los pesos



donde 0 < i < n y  es la tasa de aprendizaje

5. Repetir los pasos 2 a 4 hasta que las salidas reales y las deseadas sean iguales para todos los vectores del conjunto de entrenamiento

Siguiendo este método se garantiza que, para un conjunto de entrenamiento adecuado, después de un número finito de iteraciones el error se reduce a niveles aceptables. El número de iteraciones necesarias y el nivel de error deseado depende de cada problema particular.

**APLICACIONES**

**Asociación de patrones**: se puede aplicar a este tipo de problemas siempre que los patrones sean linealmente separables. En el campo del procesamiento de señales:

**Filtros de ruido**: Limpiar ruido de señales transmisoras de información.

**Filtros adaptativos**: Un adaline es capaz de predecir el valor de una señal en el instante t+1 si se conoce el valor de la misma en los p instantes anteriores (p es >0 y su valor depende del problema). El error de la predicción será mayor o menor según qué señal queramos predecir. Si la señal se corresponde a una serie temporal el Adaline, pasado un tiempo, será capaz de dar predicciones exactas.

Se pueden combinar múltiples Adalines formando lo que se denomina el Madaline.

**7- Redes de autopropagacion (Backpropagation)**

En 1986, Rumelhart, hinton y Williams formalizaron un método para que una red neuronal aprendiera la relación que existe entre los patrones de entrada a la red y las salidas correspondientes, utilizando más niveles de neuronas que los que utilizo rosemblatt para desarrollar el perceptron.

La red back-progagation está basada en la generalización de la regla delta, al igual que el perceptron, adaline y madaline.

La red back-propagation se caracteriza por tener una arquitectura en niveles y conexiones estrictamente hacia adelante entre las neuronas.

Utilizan aprendizaje supervisado.

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. Hay distintas versiones o reglas del algoritmo de retropropagación y distintos arquitecturas conexionistas a las que pueden ser aplicados. Durante mucho tiempo no se dispuso de algoritmos para entrenar redes multicapa, y como las redes de una capa estaban muy limitadas en cuanto a lo que eran capaces de representar, el campo de las redes neuronales artificiales estaba estancado. La invención y perfeccionamiento del algoritmo de retropropagación dio un gran impulso al desarrollo de este campo. Tiene un buen fundamento matemático y a pesar de sus limitaciones ha expandido enormemente el rango de problemas donde se aplican las redes neuronales artificiales.

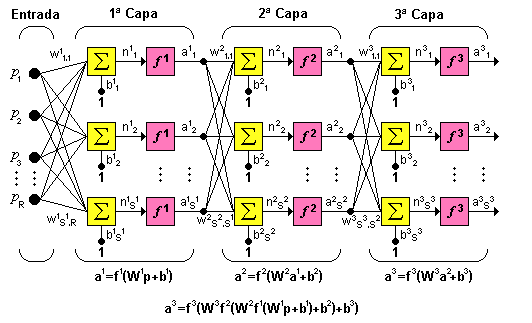
FASES:

PRIMERO: Se aplica un patrón de entrada como estímulo para la primera capa de las neuronas de la red, se va propagando a través de todas las capas superiores hasta generar una salida.

Después se compara el resultado obtenido en las neuronas de salida con la salida que se desea obtener y se calcula un valor del error para cada neurona de salida.

SEGUNDO: Estos errores se transmiten hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa intermedia que contribuyan directamente a la salida, recibiendo el porcentaje de error aproximado a la participación de la neurona intermedia en la salida original.

8- **El Algoritmo Blackprogramation en imágenes**



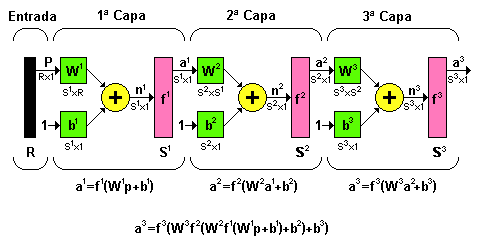
Puede notarse que esta red de tres capas equivale a tener tres redes tipo Perceptrón en cascada; la salida de la primera red, es la entrada a la segunda y la salida de la segunda red es la entrada a la tercera. Cada capa puede tener diferente número de neuronas, e incluso distinta función de transferencia.

En la figura, W1 representa la matriz de pesos para la primera capa, W2 los pesos de la segunda y así similarmente para todas las capas que incluya una red. Para identificar la estructura de una red multicapa, se empleará una notación abreviada, donde el número de entradas va seguido del número de neuronas en cada capa:

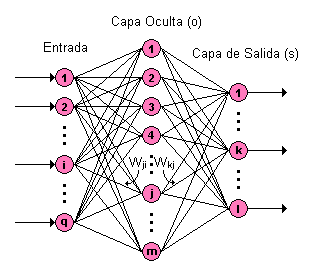
R : S1 : S2 : S3

Donde S representa el número de neuronas y el exponente representa la capa a la cual la neurona corresponde.

La notación de la figura es bastante clara cuando se desea conocer la estructura detallada de la red, e identificar cada una de las conexiones, pero cuando la red es muy grande, el proceso de conexión se torna muy complejo y es bastante útil utilizar el esquema de la siguiente figura Notación compacta de una red de tres capas



La deducción matemática de este procedimiento se realizará para una red con una capa de entrada, una capa oculta y una capa de salida y luego se generalizará para redes que tengan más de una capa oculta.



http://html.rincondelvago.com/redes-neuronales-artificiales\_1.html

**10-Teorema de Kolmogorov**

Teorema de Kolmogorov Dada cualquier función ,

f puede ser implementada exactamente por una red neuronal de tres capas sin retroalimentación que tiene una capa de entrada de n elementos que unicamente copian las entradas a la siguiente capa, (2n + 1) elementos de procesamiento en la capa intermedia y me elementos de procesamiento en la capa de salida