**REDES NEURONALES**

**SANDRA DONAY SALAZAR ARCILA**

**INTELIGENCIA ARTIFICIAL**

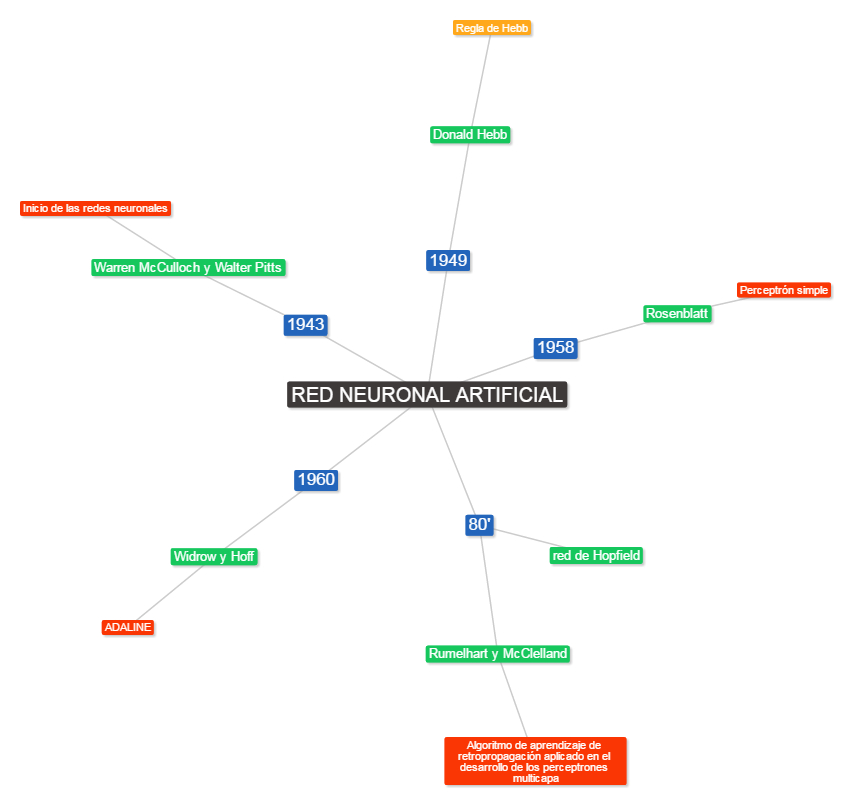
**TECNOLOGIA EN SISTEMAS DE INFORMACION**

**CARLOS LONDOÑO**

**CORPORACION DE ESTUDIOS TECNOLOGICOS DEL NORTE DEL VALLE**

**CARTAGO 5 DEL 2016**

**HISTORIA DE LAS REDES NEURONALES**

****

**VENTAJAS DE LAS REDES NEURONALES**

* **APRENDIZAJE:** Las redes neuronales tiene la habilidad de aprender mediante una etapa que se llama etapa de aprendizaje. Esto consiste en proporcionar a la red neuronal datos como entradas a su vez que se le indica cuál es la salida (respuesta) esperada.
* **AUTO ORGANIZACIÓN:** Una red neuronal crea su propia representación de la información en su interior, descargando el usuario de esta tarea.
* **TOLERANCIA A FALLOS:** Debido a que una red neuronal almacena información de forma redundante, es a puede seguir respondiendo de manera aceptable aun si se daña parcialmente.
* **FLEXIBILIDAD**: Una red neuronal 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.
* **NO SON ALGORITMICAS:** La gran diferencia del empleo de las redes neuronales en relación con otras aplicaciones de la computación radica en que no son algorítmicas; esto es, no se programan haciéndoles seguir una secuencia predefinida de instrucciones. Las RNA generan ellas mismas sus propias "reglas", para asociar la respuesta a su entrada; es decir, aprende por ejemplos y de sus propios errores.

El conocimiento de una RNA se encuentra en la función de activación utilizada y en los valores de sus pesos.

**DESVENAJAS DE LAS REDES NEURONALES**

* **COMPLEJIDAD:** Complejidad de aprendizaje para grandes tareas, cuanto más cosas se necesiten que aprenda una red, más complicado será enseñarle.
* **TIEMPO DE APRENDIZAJE:** 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.
* **ENTRENAMIENTO:** 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.
* **FALTA DE REGLAS:** reglas definitorias que ayuden a construir una red para un problema dado, hay muchos factores a tomar en cuanta: el algoritmo de aprendizaje, la arquitectura, el número de neuronas por capa, el número de capas, la representación de los datos y mucho más y otro de los problemas es la llamada  caja negra el problema es que cuando modelamos estadísticamente somos capaces de ver que variables forman parte del modelo o cuales de las que finalmente se utilizaron para modelar fueron seleccionadas por los algoritmos para predecir o clasificar, podemos ver sus pesos y la ecuación final, cosa que no es posible en las redes neuronales.
* **HARDWARE**: las máquinas hoy en día son serie - sólo ejecutan una instrucción a la vez. por ello, modelar procesos paralelos en máquinas serie puede ser un proceso que consuma mucho tiempo.

**APLICACIONES DE LAS REDES NEURONALES**

* Clasificaciones de patrones
* Aproximación de funciones
* Mapeos
* Reconocimiento del habla
* Reconocimiento de textos manuscritos
* Identificación de blancos de radares
* Simulación de centrales de producción de entrega
* Operación en tiempo real
* Otro tipo especial de redes neuronales artificiales se ha aplicado en conjunción con los [algoritmos genéticos](https://es.wikipedia.org/wiki/Algoritmo_gen%C3%A9tico) (AG) para crear controladores para [robots](https://es.wikipedia.org/wiki/Robot).
* Procesadores digitales de señales

**FUNCIONES DE ACTIVACIÓN**

La Función de Activación de un nodo define la salida de un nodo dada una entrada o un conjunto de entradas. Se podría decir que un circuito estándar de computador se comporta como una red digital de funciones de activación al activarse como "ON" (1) u "OFF" (0), dependiendo de la entrada. Esto es similar al funcionamiento de un [Perceptrón](https://es.wikipedia.org/wiki/Perceptr%C3%B3n" \o "Perceptrón) en una [Red neuronal artificial](https://es.wikipedia.org/wiki/Red_neuronal_artificial).

En las redes neurales inspiradas sobre la biología, la función de activación es usualmente una abstracción representando una tasa de potencial de activación gatillándose en la celda. En su forma simplificada, esta función es binaria, esto es, se activa la neurona o no.

**TIPOS DE FUNCION DE ACTIVACION**

**Funciones de activación**

* Función de Activación Escalón
* Función de Activación Identidad
* Función de Activación Lineal -Mixta
* Función de Activación Sigmoidal
* Función de transferencia gaussiana
* Funciones de saturación

**Función de activación Escalón:** La función escalón se asocia a neuronas binarias en las cuales cuando la suma de las entradas es mayor o igual que el umbral de la neurona, la activación es 1, si es menor, la activación es 0 (ó –1). Las redes formadas por este tipo de neuronas son fáciles de implementar en hardware, pero sus capacidades están limitadas.

**Función de activación identidad:** es adecuada cuando la función de activación que hemos utilizado para calcular la activación de la unidad es de tipo umbral.

**Función de activación Lineal – mixta:** La función escalón se asocia a neuronas binarias en las cuales cuando la suma de las entradas es mayor o igual que el umbral de la neurona, la activación es 1, si es menor, la activación es 0 (ó –1). Las redes formadas por este tipo de neuronas son fáciles de implementar en hardware, pero sus capacidades están limitadas.

**Función de activación Sigmoidal:** Cualquier función definida simplemente en un intervalo de posibles valores de entrada, con un incremento monotónico y que tengan ambos limites superiores e inferiores (por ejemplo las funciones sigmoidal y arco tangente), podrá realizar la función de activación o transferencia de forma satisfactoria

**Función de transferencia gaussiana:** Los centros y anchura de estas funciones pueden ser adaptados, lo cual las hace más adaptativas que las funciones sigmoidales.

**Función de saturación:** Son las funciones en las que los incrementos o disminuciones de la intensidad de la actividad de la unidad producen incrementos o disminuciones de los valores de salida hasta llegar a unos límites de salida máxima o mínima a partir de los cuales la salida se estabiliza y es la misma a pesar del incremento o disminución de la intensidad de actividad de la unidad.

**PERCEPTRON**

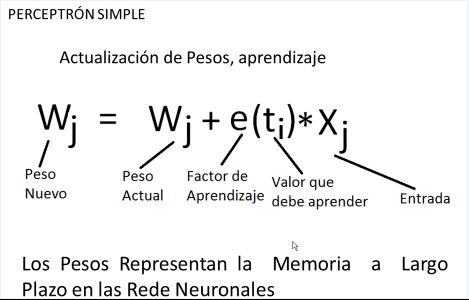
**HISTORIA**

En 1943, Warren McCulloch y Walter Pitts introdujeron una de las primeras neuronas artificiales. La característica principal de su modelo de neurona es que un suma ponderada de las señales de entrada se compara con un umbral para determinar la neurona de salida. Cuando la suma es mayor o igual al umbral, la salida es 1. Cuando la suma es menor que el umbral, la salida es 0. A finales de 1950 Frank Rosenblatt y otros investigadores desarrollaron una clase de redes neuronales llamadas perceptrones. Las neuronas de estas redes eran similares a las de McCulloch y Pitts.

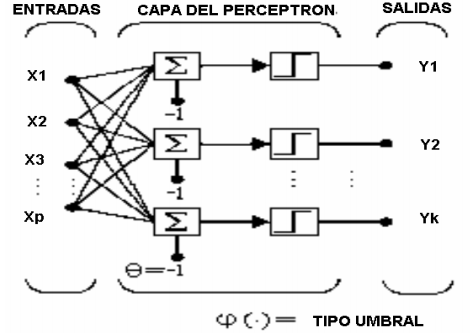
La contribución clave de Rosenblatt fue la introducción de una regla de aprendizaje para la formación de redes perceptrón para resolver problemas de reconocimiento de patrones. Demostró que su regla de aprendizaje siempre convergirá a los pesos correctos de la red, si existen pesos que solucionan el problema. El Perceptrón pudo incluso aprender cuando se inicializaba con valores aleatorios de sus pesos y bias.

El Perceptrón es limitado. Dichas limitaciones fueron publicadas en el libro Perceptrons por Marvin Minsky y Seymour Papert. Ellos demostraron que las redes perceptrón eran incapaces de implementar ciertas funciones elementales. No fue sino hasta la década de los 8O’s que estas limitaciones fueron superadas con las redes perceptrón mejoradas (multicapa) asociadas con reglas de aprendizaje.

**FORMULA MATEMATICA DEL PERCEPTRON**



**ESTRUCTURA DEL PERCEPTRON**



**USO DEL PERCEPTRON**

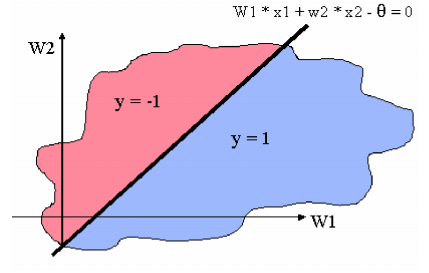
**FUNCION DE ACTIVACION DEL PERCEPTRON**

Se puede simplificar la función de activación incorporando el valor umbral en el sumatorio. Basta dañar a la red una unidad extra tal que: § Siempre tiene la salida al valor 1 § Se conecta a todas las unidades de la red § El peso de la conexión es el umbral de la neurona a la que se conecta y = g ( å -q = i n i i w x 1 ) = g ( å= n i i i w x 0 ) El origen de las entradas no es importante. Pueden proceder de otros perceptrones o de otra clase de unidades de ccmputación. La interpretación geométrica es sencilla: la superficie de separación entre las entradas que originan una respuesta 1 y las de respuesta 0, es un hiperplano que divide al espacio de entradas en dos semiespacios. La red tipo Perceptrón emplea principalmente dos funciones de transferencia, escalón con salidas 1, 0 o escalón con salidas 1, -1; su uso depende del valor de salida que se espera para la red, es decir si la salida de la red es unipolar o bipolar; sin embargo la función escalón con salidas 1, -1 es preferida sobre la escalón con salidas 1, 0 ya que el tener un cero multiplicando algunas de los valores resultantes del producto de las entradas por el vector de pesos, ocasiona que estos no se actualicen y que el aprendizaje sea más lento.

No todas las funciones lógicas pueden computarse con un perceptrón simple. Este hecho tiene que ver con la geometría del cubo n-dimensional cuyos vértices representa la combinación de los valores lógicos de los argumentos. Cada función lógica separa los vértices en dos clases: aquéllos para los que la función es l, y aquéllos para los que la función es 0. Si los puntos en los que la función vale 1 pueden separarse mediante un hiperplano de los puntos en los que la función es 0, entonces la función es computable con un perceptrón. Si esto no es así, la función no es computable. Las funciones computables con un perceptrón son las funciones linealmente separables. Un ejemplo de función lógica no linealmente separable es el XOR.

**COMO SE ENTRENA UN PERCEPTRON**

Como sabemos, un algoritmo de aprendizaje es un método adaptativo por el que una red de PE se automodifica para implementar el comportamiento deseado. Esto se hace presentando algunos ejemplos de la función entrada-salida a la red. Se presenta un ejemplo y se ejecuta una acción correctiva iterativamente hasta que la red aprende a producir la respuesta deseada. El conjunto de entrenamiento es el conjunto de los ejemplos de los que la red va a aprender. Usamos la siguiente notación: El vector de entrada al perceptron es x = (x1,..., xn) . Si los pesos son los valores reales w1,..., wn y el umbral es U, diremos que w = (w1, ..., wn, wn+1) con wn+1 = – U es el vector extendido de Teorema de convergencia Si las clases son linealmente separables, el algoritmo del perceptrón converge a una solución correcta en un número finito de pasos para cualquier elección inicial de pesos pesos del perceptrón y que (x1,..., xn, 1) es el vector extendido de entradas (se añade la entrada de tendencia o bias con valor 1 fijo). La computación de un perceptrón puede expresarse mediante un producto escalar, ya que ƶi wi xi U es equivalente a w • x  0 donde w y x son los vectores extendidos de pesos y entradas, respectivamente. La convergencia del algoritmo de aprendizaje del perceptrón se basa en que cada perceptrón realiza la comprobación w • x > 0, ó w • x  0, pero son equivalentes cuando el conjunto de entrenamiento es finito, lo cual siempre es cierto en problemas prácticos. Una forma habitual de comenzar el algoritmo de entrenamiento es inicializando aleatoriamente los pesos de la red y mejorar los parámetros iniciales, comprobando a cada paso, si puede lograrse una separación mejor del conjunto de entrenamiento. Cada vector de pesos w define un hiperplano que separa los puntos con salida 1 (w • x  0) de los puntos con salida 0 (w • x 0). Interpretación gráfica o geométrica del perceptrón La ecuación de evaluación de las salidas representa un plano (un hiperplano en dimensión n) que divide el espacio en dos partes. El entrenamiento consiste en buscar el plano que hace la división adecuada, para que los puntos cuya salida es 1 queden todos en la misma región, y los que tienen salida esperada 0, queden todos en la otra región. Sean P y N dos conjuntos finitos de puntos en Rn que queremos separar linealmente. Se desea encontrar un vector w tal que su hiperplano asociado separe los dos conjuntos (salida 1 para P y 0 para N). El error de un perceptron con un vector de pesos w es el número de puntos incorrectamente clasificados. El algoritmo de entrenamiento debe minimizar este error E(w). Una de las posibles estrategias es la de usar un algoritmo voraz local que calcule el error del perceptrón para un vector de pesos dado, buscando una dirección en el espacio de pesos en la que moverse y actualizandolos, seleccionando para ello nuevos valores de acuerdo con la dirección de búsqueda. E



**DONDE SE EVIDENCIA EL USO DEL PERCEPTRON**

**Aquí falta**

**ADALINE**

**HISTORIA DEL ADALINE**

Los primeros modelos de redes neuronales datan de [1943](https://es.wikipedia.org/wiki/1943) por los neurólogos Warren McCulloch y Walter Pitts. Años más tarde, en 1949, Donald Hebb desarrolló sus ideas sobre el aprendizaje neuronal, quedando reflejado en la "regla de Hebb". En 1958, Rosenblatt desarrolló el [perceptrón](https://es.wikipedia.org/wiki/Perceptr%C3%B3n" \o "Perceptrón) simple, y en 1960, Widrow y Hoff desarrollaron el [ADALINE](https://es.wikipedia.org/wiki/Adaline), que fue la primera aplicación industrial real.

En los años siguientes, se redujo la investigación, debido a la falta de modelos de aprendizaje y el estudio de Minsky y Papert sobre las limitaciones del perceptrón. Sin embargo, en los años 80, volvieron a resurgir las RNA gracias al desarrollo de la red de Hopfield, y en especial, al algoritmo de aprendizaje de retropropagación (BackPropagation) ideado por Rumelhart y McClelland en 1986 que fue aplicado en el desarrollo de los perceptrones multicapa.

**FORMULA MATEMATICA DEL ADALINE**

La regla Delta utiliza la diferencia entre la salida producida para cada patrón y la deseada

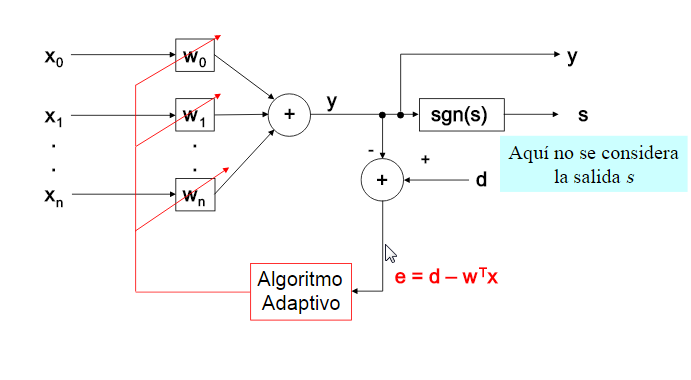
Se calcula una función de error para todo el conjunto de patrones:

regla delta

Donde E es el error global y Epel error cuadrático por patrón

Esta regla busca el conjunto de pesos que minimiza la función de error, esto se realiza mediante un proceso iterativo donde se presentan los patrones y se van modificando los parámetros de la red mediante la regla del descenso del gradiente

**ESTRUCTURA DEL ADALINE**

****

**FUNCION DE ACTIVACION DE ADALINE**

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 {\displaystyle x} de entrada que pertenece al espacio {\displaystyle R^{m}}.
* Por cada neurona, existe un vector {\displaystyle w} de pesos sinápticos que indican la fuerza de conexión entre los valores de entrada y la neurona. En la práctica representan la[ponderación](https://es.wikipedia.org/w/index.php?title=Ponderaci%C3%B3n&action=edit&redlink=1) de cada entrada sobre la neurona.
* Una constante {\displaystyle \theta }  
  .
* La salida {\displaystyle y}***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 }

**ENTRENAMIENTO DE UNA RED ADALINE**

A diferencia del perceptrón, a la hora de modificar los pesos durante el entrenamiento, el Adaline tiene en cuenta el grado de corrección de la salida estimada respecto a la deseada.[2](https://es.wikipedia.org/wiki/Adaline#cite_note-aprendizaje-2) Esto se consigue mediante la aplicación de la *regla Delta*, y que se define, para un patrón de entrada {\displaystyle x^{p}}  con una salida estimada  {\displaystyle y^{p}} y una salida deseada  {\displaystyle d^{p}} , como  {\displaystyle |d^{p}-y^{p}|} .

Dado que el objetivo del Adaline es poder estimar de la manera más exacta la salida (conseguir una salida exacta es prácticamente imposible en la mayoría de los casos), se busca minimizar la desviación de la red para todos los patrones de entrada, eligiendo una medida del error global. Normalmente se utiliza el error cuadrático medio.

{\displaystyle E={\frac {1}{2}}\sum \_{p=1}^{m}(d^{p}-y^{p})^{2}} 

La manera de reducir este error global es ir modificando los valores de los pesos al procesar cada entrada, de forma iterativa, mediante la regla del [descenso del gradiente](https://es.wikipedia.org/w/index.php?title=Descenso_del_gradiente&action=edit&redlink=1). Suponiendo que tenemos una constante de aprendizaje {\displaystyle \alpha } :

{\displaystyle \Delta \_{p}w\_{j}=-\alpha {\frac {\partial E^{p}}{\partial w\_{j}}}} 

Si operamos con la derivada, queda:

{\displaystyle \Delta \_{p}w\_{j}=\alpha (d^{p}-y^{p})\cdot x\_{j}} 

Que será la expresión que utilizaremos por cada entrada para modificar los pesos.

**USO DEL ADALINE**

La principal aplicación de las redes tipo adaline se encuentran en el campo de procesamiento de señales, concretamente en el diseño de filtros capaces de eliminar ruido en señales portadores de información

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

**RED RETROPOPAGACION (BACKPROPAGATION)**

**QUE ES**

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.

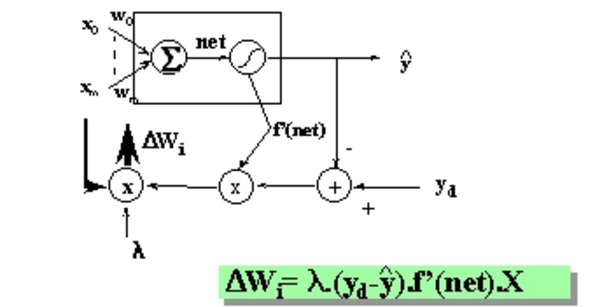
**USO**

La propagación hacia atrás de errores o retropropagación (del inglés backpropagation) es un [algoritmo](https://es.wikipedia.org/wiki/Algoritmo) de [aprendizaje supervisado](https://es.wikipedia.org/wiki/Aprendizaje_supervisado) que se usa para entrenar [redes neuronales artificiales](https://es.wikipedia.org/wiki/Redes_neuronales_artificiales). El algoritmo emplea un ciclo propagación – adaptación de dos fases. Una vez que se ha aplicado un patrón a la entrada de la red como estímulo, este se propaga desde la primera capa a través de las capas superiores de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas.

Las salidas de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo las neuronas de la capa oculta solo reciben una fracción de la señal total del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total.

La importancia de este proceso consiste en que, a medida que se entrena la red, las neuronas de las capas intermedias se organizan a sí mismas de tal modo que las distintas neuronas aprenden a reconocer distintas características del espacio total de entrada. Después del entrenamiento, cuando se les presente un patrón arbitrario de entrada que contenga ruido o que esté incompleto, las neuronas de la capa oculta de la red responderán con una salida activa si la nueva entrada contiene un patrón que se asemeje a aquella característica que las neuronas individuales hayan aprendido a reconocer durante su entrenamiento.

**ALGORITMO BACKPROPAGATION**

****

**TEOREMA DE KOLMOGOROV**

El **teorema de Kolmogórov–Arnold–Moser** o **teorema KAM** es un resultado de [sistemas dinámicos](https://es.wikipedia.org/wiki/Sistemas_din%C3%A1micos) sobre la persistencia de [movimientos cuasiperiódicos](https://es.wikipedia.org/wiki/Movimiento_cuasiperi%C3%B3dico). Este teorema resuelve parcialmente el [problema de los divisores pequeños](https://es.wikipedia.org/w/index.php?title=Problema_de_los_divisores_peque%C3%B1os&action=edit&redlink=1) (que origina problemas de convergencia en sistemas con múltiples frecuencias). El teorema explica cómo se modifica el aspecto de las trayectorias de un sistema integrable bajo pequeñas perturbaciones.

El movimiento en un [sistema integrable](https://es.wikipedia.org/wiki/Sistema_integrable) está confinado a una [hiper]superficie [toroidal](https://es.wikipedia.org/wiki/Toro_(geometr%C3%ADa)" \l "El_toro_en_n_dimensiones" \o "Toro (geometría)). Diferentes condiciones iniciales del sistema originan diferentes toros en el [espacio fásico](https://es.wikipedia.org/wiki/Espacio_f%C3%A1sico). Que las trayectorias de un sistema integrable de dimensión *n* están confinadas a hipersuperficies de tipo {\displaystyle \mathbb {T} ^{n}} pueden deducirse del tratamiento de las [variables acción-ángulo](https://es.wikipedia.org/wiki/Sistema_hamiltoniano_integrable#Sistema_que_admite_n_variables_.C3.A1ngulo-acci.C3.B3n), al existir *n* variables "ángulo" periódicas.

El teorema KAM establece que, si un sistema está sometido a una pequeña perturbación no lineal, algunos toros serán deformados y otros destruidos. Los que sobreviven son aquellos que tienen un cociente de frecuencias suficientemente irracional. Es decir, se destruyen aquellos cuyo cociente de frecuencias se acerca más a un número racional, dados por la relación

{\displaystyle \left\vert {\frac {\omega \_{2}}{\omega \_{1}}}-{\frac {m}{s}}\right\vert >{\frac {k(\epsilon )}{\sqrt {s}}}}Con {\displaystyle k(\epsilon \rightarrow 0)\rightarrow 0}. El último toro en destruirse es el más irracional de todos (el que guarda mayor semejanza con el [número áureo](https://es.wikipedia.org/wiki/N%C3%BAmero_%C3%A1ureo)). Informalmente el teorema establece que:

"Para perturbaciones suficientemente pequeñas, casi todos los toros invariantes se preservan [en el sistema perturbado] (excluyendo aquellos con vectores de frecuencia racionales)