**PRINCIPLES OF TRAINING MULTI-LAYER NEURAL NETWORK USING BACKPROPAGATION**

|  |
| --- |
| The project describes teaching process of multi-layer neural network employing *backpropagation* algorithm. To illustrate this process the three layer neural network with two inputs and one output,which is shown in the picture below, is used:  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img01.gif  Each neuron is composed of two units. First unit adds products of weights coefficients and input signals. The second unit realise nonlinear function, called neuron activation function. Signal *e* is adder output signal, and *y = f(e)* is output signal of nonlinear element. Signal *y* is also output signal of neuron.  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img01b.gif  To teach the neural network we need training data set. The training data set consists of input signals (*x1* and *x2* ) assigned with corresponding target (desired output) *z*. The network training is an iterative process. In each iteration weights coefficients of nodes are modified using new data from training data set. Modification is calculated using algorithm described below: Each teaching step starts with forcing both input signals from training set. After this stage we can determine output signals values for each neuron in each network layer. Pictures below illustrate how signal is propagating through the network, Symbols *w(xm)n* represent weights of connections between network input *xm* and neuron *n* in input layer. Symbols *yn* represents output signal of neuron *n*.  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img02.gif  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img03.gif  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img04.gif  Propagation of signals through the hidden layer. Symbols *wmn* represent weights of connections between output of neuron *m* and input of neuron *n* in the next layer.  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img05.gif  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img06.gif  Propagation of signals through the output layer.  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img07.gif  In the next algorithm step the output signal of the network *y* is compared with the desired output value (the target), which is found in training data set. The difference is called error signal *d* of output layer neuron.  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img08.gif  It is impossible to compute error signal for internal neurons directly, because output values of these neurons are unknown. For many years the effective method for training multiplayer networks has been unknown. Only in the middle eighties the backpropagation algorithm has been worked out. The idea is to propagate error signal *d* (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img09.gif  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img10.gif  The weights' coefficients *wmn* used to propagate errors back are equal to this used during computing output value. Only the direction of data flow is changed (signals are propagated from output to inputs one after the other). This technique is used for all network layers. If propagated errors came from few neurons they are added. The illustration is below:  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img11.gif  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img12.gif  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img13.gif  When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas below *df(e)/de* represents derivative of neuron activation function (which weights are modified).  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img14.gif  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img15.gif  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img16.gif  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img17.gif  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img18.gif  http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img19.gif  Coefficient *h* affects network teaching speed. There are a few techniques to select this parameter. The first method is to start teaching process with large value of the parameter. While weights coefficients are being established the parameter is being decreased gradually. The second, more complicated, method starts teaching with small parameter value. During the teaching process the parameter is being increased when the teaching is advanced and then decreased again in the final stage. Starting teaching process with low parameter value enables to determine weights coefficients signs.   **References** Ryszard Tadeusiewcz "Sieci neuronowe", Kraków 1992 |

**La propagación hacia atrás de errores** o **retropropagación** (del inglés *backpropagation*) es un [algoritmo](http://es.wikipedia.org/wiki/Algoritmo) de [aprendizaje supervisado](http://es.wikipedia.org/wiki/Aprendizaje_supervisado) que se usa para entrenar [redes neuronales artificiales](http://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.

## Estructura del Perceptrón

El modelo de una neurona artificial es una imitación del proceso de una neurona biológica, tal como se ve en la figura: [Archivo:Perceptrón.png](http://commons.wikimedia.org/wiki/Special:UploadWizard?uselang=es&wpDestFile=Perceptr%C3%B3n.png" \o "Archivo:Perceptrón.png)

Donde los p_i son las entradas (por las dendritas) a la neurona, estas sufren un efecto multiplicador w_i por la comunicación de las mismas al núcleo de la neurona, donde se sumaran mediante:

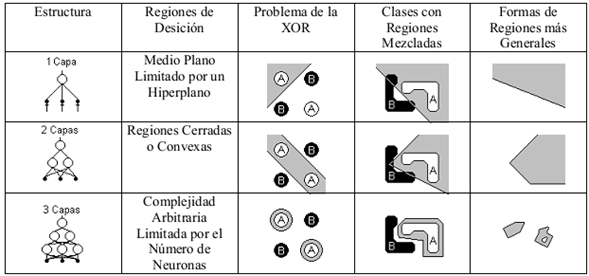
n=\sum^{i=q}_{i=1}(w_i*p_i)+b

La salida de la neurona luego es modificada mediante la función de transferencia f:

a=f(n)=f(\sum^{i=q}_{i=0}(p_i*w_i)+b)

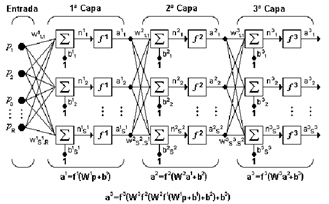
## Perceptrón Multicapa

Un Perceptrón multicapa es una red con alimentación hacia delante, compuesta de varias capas de neuronas entre la entrada y la salida de la misma, esta red permite establecer regiones de decisión mucho más complejas que las de dos semiplanos, como lo hace el Perceptrón de un solo nivel. Las capacidades del Perceptrón multicapa con dos y tres capas y con una única neurona en la capa de salida se muestran en la siguiente figura:

[](http://commons.wikimedia.org/wiki/File:Perceptr%C3%B3nMulticapa.png)

Perceptrón simple, activándose su salida para los patrones de un lado del hiperplano, si el valor de los pesos de las conexiones entre las neuronas de la segunda capa y una neurona del nivel de salida son todos igual a 1, y la función de salida es de tipo hardlim, la salida de la red se activará sólo si las salidas de todos los nodos de la segunda capa están activos, esto equivale a ejecutar la función lógica AND en el nodo de salida, resultando una región de decisión intersección de todos los semiplanos formados en el nivel anterior. La región de decisión resultante de la intersección será una región convexa con un número de lados a lo sumo igual al número de neuronas de la segunda capa. A partir de este análisis surge el interrogante respecto a los criterios de selección para las neuronas de las capas ocultas de una red multicapa, este número en general debe ser lo suficientemente grande como para que se forme una región compleja que pueda resolver el problema, sin embargo no debe ser muy grande pues la estimación de los pesos puede ser no confiable para el conjunto de los patrones de entrada disponibles. Hasta el momento no hay un criterio establecido para determinar la configuración de la red y esto depende

Estructura de la Red[[editar](http://es.wikipedia.org/w/index.php?title=Propagaci%C3%B3n_hacia_atr%C3%A1s&action=edit&section=3" \o "Editar sección: Estructura de la Red)]

[](http://commons.wikimedia.org/wiki/File:EstructuraRedNeuronal.png)

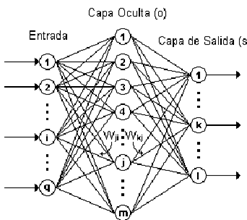
Puede notarse que esta red de tres capas equivale a tener tres redes tipo Perceptrón en cascada; la salida de la primera red, es la entrada a la segunda y la salida de la segunda red es la entrada a la tercera. Cada capa puede tener diferente número de neuronas, e incluso distinta función de transferencia.

Regla de Aprendizaje

La red Backpropagation trabaja bajo aprendizaje supervisado y por tanto necesita un set de entrenamiento que le describa cada salida y su valor de salida esperado de la siguiente forma:

{(p_1,t_1)},{(p_2,t_2)},...,{(p_q,t_q)}   (1)

Dondep_q es una entrada a la red y t_q es la correspondiente salida deseada para el patrón q-ésimo. El algoritmo debe ajustar los parámetros de la red para minimizar el error cuadrático medio. Es importante recalcar que no existe una técnica para determinar el número de capas ocultas, ni el número de neuronas que debe contener cada una de ellas para un problema específico, esta elección es determinada por la experiencia del diseñador, el cual debe cumplir con las limitaciones de tipo computacional. Cada patrón de entrenamiento se propaga a través de la red y sus parámetros para producir una respuesta en la capa de salida, la cual se compara con los patrones objetivo o salidas deseadas para calcular el error en el aprendizaje, este error marca el camino mas adecuado para la actualización de los pesos y ganancias que al final del entrenamiento producirán una respuesta satisfactoria a todos los patrones de entrenamiento, esto se logra minimizando el error cuadrático medio en cada iteración del proceso de aprendizaje.

[](http://commons.wikimedia.org/wiki/File:CapaOculta.png)

q: Número de componentes del vector de entrada.

m: Número de neuronas de la capa oculta.

l: Número de neuronas de la capa de salida.

Para iniciar el entrenamiento se le presenta a la red un patrón de entrenamiento, el cual tiene q componentes como se describe en la ecuación (2):


   p =
   \begin{bmatrix}
      p_1 \\
      p_2 \\
      . \\
      . \\
      p_i \\
      . \\
      . \\
      p_q
   \end{bmatrix} (2)


Cuando se le presenta a la red un patrón de entrenamiento, este se propaga a través de las conexiones existentes produciendo una entrada neta n en cada una las neuronas de la siguiente capa, la entrada neta a la neurona j de la siguiente capa debido a la presencia de un patrón de entrenamiento en la entrada está dada por la ecuación (3), nótese que la entrada neta es el valor justo antes de pasar por la función de transferencia:

n^{0}_j=\sum^{q}_{i=1}(w^{0}_{ji}*p_i)+b  (3)

w^{0}_{ji}:Peso que une la componente i de la entrada con la neurona j de la capa oculta.

p_i:Componente i del vector p que conteiene el patrón de entrenamiento de q componentes.

b^0:Ganancia de la neurona j de la capa oculta.

Donde (0) representa la capa a la que pertenece cada parámetro, es este caso la capa oculta. Cada una de las neuronas de la capa oculta tiene como salida a_j^0, dada por:

a_j^0=f^0 (\sum^{q}_{i=1}(w_{ji}^0*p_i)+b_j^0 )  (4)

f^0: Función de transferencia de las neuronas de la capa oculta. Las salidas a_j^0 de las neuronas de la capa oculta (de m componentes) son las entradas a los pesos de conexión de la capa de salida, este comportamiento esta descripto por:

n_k^s = \sum^{m}_{j=1}(w_{kj}^s*a_j^0)+b_k^s  (5)

w_{kj}^s: Peso que une la neurona j de la capa oculta con la neurona k de la capa de salida, la cual cuenta con s neuronas.

a_j^0: Salida de la neurona j de la capa oculta, la cual cuenta con m neuronas.

b_k^s: Ganancia de la neurona k de la capa de salida.

n_k^s: Entrada neta a la neurona k de la capa de salida.

La red produce una salida final: a_k^s = f^s(n^s_k)  (6)

f^s: Función de transferencia de las neuronas de la capa de salida.

La salida de la red de cada neurona a_k^s se compara con la salida deseada t_k para calcular el error en cada unidad: \delta_k=(t_k-a_k^s)

El error debido a cada patrón p propagado está dado por:

ep^2=1/2  \sum^{s}_{k=1}(\delta_k^2)

ep^2: Error cuadrático medio para cada patrón de entrada p.

\delta_k: Error en la neurona k de la capa de salida con l neuronas.

Este proceso se repite para el número total de patrones de entrenamiento (r), para un proceso de aprendizaje exitoso el objetivo del algoritmo es actualizar todos los pesos y ganancias de la red minimizando el error cuadrático medio total descrito en:

e^2= \sum^{r}_{p=1}(ep^2)

e^2: Error total en el proceso de aprendizaje en una iteración luego de haber presentado a la red los r patrones de entrenamiento. El error que genera una red neuronal en función de sus pesos, genera un espacio de n dimensiones, donde n es el número de pesos de conexión de la red, al evaluar el gradiente del error en un punto de esta superficie se obtendrá la dirección en la cual la función del error tendrá un mayor crecimiento, como el objetivo del proceso de aprendizaje es minimizar el error

[Archivo:ErrorBackpropagation.png](http://commons.wikimedia.org/wiki/Special:UploadWizard?uselang=es&wpDestFile=ErrorBackpropagation.png)

debe tomarse la dirección negativa del gradiente para obtener el mayor decremento del error y de esta forma su minimización, condición requerida para realizar la actualización de la matriz de pesos en el algoritmo Backpropagation:

(Continua)

Minimización del Error

Los algoritmos en [Aprendizaje Automático](http://es.wikipedia.org/wiki/Aprendizaje_Autom%C3%A1tico) pueden ser clasificados en dos categorías: supervisados y no supervisados. Los algoritmos en aprendizaje supervisado son usados para construir "modelos" que generalmente predicen ciertos valores deseados. Para ello, los algoritmos supervisados requieren que se especifiquen los valores de salida (*output*) u objetivo (*target*) que se asocian a ciertos valores de entrada (*input*). Ejemplos de objetivos pueden ser valores que indican éxito/fallo, venta/no-venta, pérdida/ganancia, o bien ciertos atributos multi-clase como cierta gama de colores o las letras del alfabeto. El conocer los valores de salida deseados permite determinar la calidad de la aproximación del modelo obtenido por el algoritmo.

La especificación de los valores entrada/salida se realiza con un conjunto consistente en pares de vectores con entradas reales de la forma (\boldsymbol x,\boldsymbol t), conocido como conjunto de entrenamiento o conjunto de ejemplos, donde (\boldsymbol x) serán nuestros parámetros de entrada y (\boldsymbol t) los de salida de la red. Los algoritmos de aprendizaje generalmente calculan los parámetros \boldsymbol W de una función N(\boldsymbol x;\boldsymbol W) que permiten aproximar los valores de salida en el conjunto de entrenamiento.

Si (\boldsymbol x^{(q)},\boldsymbol t^{(q)}), q=1,\ldots,p, son los elementos del conjunto de entrenamiento, la calidad de la aproximación en el ejemplo q se puede medir a través del error cuadrático:


E(\boldsymbol x^{(q)}; \boldsymbol W)=\frac{1}{2} \|N(\boldsymbol x^{(q)};\boldsymbol W) -\boldsymbol t^{(q)}\|^2
,

donde \|\cdot\| es la norma euclidiana.

El error total es la suma de los errores de los ejemplos:

E(\boldsymbol W)=\sum_{q=1}^p E(\boldsymbol x^{(q)}; \boldsymbol W).

Un método general para minimizar el error es el actualizar los parámeros de manera iterativa. El valor nuevo de los parámetros se calcula al sumar un incremento \Delta\boldsymbol W al valor actual:


\boldsymbol W := \boldsymbol W + \Delta\boldsymbol W


El algoritmo se detiene cuando \boldsymbol W converge o bien cuado el error alcanza un mínimo valor deseado.

Si la función N(\boldsymbol x;\boldsymbol W) usada para aproximar los valores de salida es diferenciable respecto a los parámetros \boldsymbol W, podemos usar como algoritmo de aprendijaze el método de gradiende descendiente. En este caso, el incremento de los parámetros se expresa como

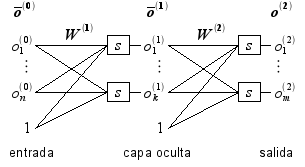

\Delta\boldsymbol W=-\gamma\frac{\partial E(\boldsymbol W)}{\partial \boldsymbol W},


donde 0<\gamma<1 es un parámetro conocido como factor de aprendizaje.

Antes de continuar introduciremos un poco de notación. Definimos \bar{\boldsymbol v}=(v_1,\ldots,v_n,1)^T como el vector extendido del vector \boldsymbol v=(v_1,\ldots,v_n)^T. El par (\boldsymbol x, \boldsymbol t)representará a un elemento del conjunto de entrenamiento y una relación de entrada-salida, a menos que se indique otra cosa.

Red Neuronal con una Capa Oculta[[editar](http://es.wikipedia.org/w/index.php?title=Propagaci%C3%B3n_hacia_atr%C3%A1s&action=edit&section=6" \o "Editar sección: Red Neuronal con una Capa Oculta)]

La función la usaremos para aproximar los valores de salida de una red neuronal artificial con una capa oculta. La red está constituida por una capa de entrada (*input layer*), una capa oculta (*hidden layer*) y una capa de salida (*output layer*), tal como se ilustra con la siguiente figura:

[](http://commons.wikimedia.org/wiki/File:RedNeuronal.png)

Los elementos que constituyen la red neuronal son los siguientes:

* s es una función de valores reales, conocida como la función de transferencia.
* \bar{\boldsymbol o}^{(0)} es la capa de entrada, considerado como el vector extendido del ejemplo \boldsymbol o^{(0)}=\boldsymbol x=(x_1,\ldots,x_n)^T.
* \bar{\boldsymbol o}^{(1)} es la capa oculta, el vector extendido de \boldsymbol o^{(1)}=(o^{(1)}_1,\ldots,o^{(1)}_k)^T.
* \boldsymbol o^{(2)}=(o_1,\ldots,o_m)^T es la capa de salida, considerado como el vector que aproxima al valor deseado \boldsymbol t=(t_1,\ldots,t_m)^T.
* \boldsymbol W^{(1)} es una matriz de tamaño (n+1)\times k cuyos valores W^{(1)}_{ij} son los pesos de la conexión entre las unidades \bar o^{(0)}_i y o^{(1)}_j.
* \boldsymbol W^{(2)} es una matriz de tamaño (k+1)\times m cuyos valores W^{(2)}_{ij} son los pesos de la conexión entre las unidades \bar o^{(1)}_i y o^{(2)}_j.

De estos elementos, únicamente las matrices \boldsymbol W^{(l)} son consideradas como los parámetros de la red, ya que los valores \bar{\boldsymbol o}^{(l)} son el resultado de cálculos que dependen de las matrices de pesos, del valor de entrada \bar{\boldsymbol x} y de la función de transferencia s.

La función de transferencia s que consideraremos en nuestro algoritmo es conocida como función sigmoidal, y esta definida como


s(u)=\frac{1}{1+\exp(-u)}


esta función además de ser diferenciable, tiene la particularidad de que su derivada se puede expresar en términos de sí misma:


\frac{d s(u)}{du}=s(u)(1-s(u)).


esto nos servirá para simplificar los cálculos en el algoritmo de aprendizaje aquí descrito.

Descripción del Algoritmo

A grandes rasgos:

1. Calcular la salida de la red \boldsymbol o^{(2)} a partir de uno de los conjuntos de valores de prueba x.
2. Comparar con la salida correcta t y calcular el error según la fórmula:

E(\boldsymbol x; \boldsymbol W^{(1)},\boldsymbol W^{(2)})=\frac{1}{2}\sum^{m}_{i=1} (t_{i}-o^{(2)}_{i})^{2}.

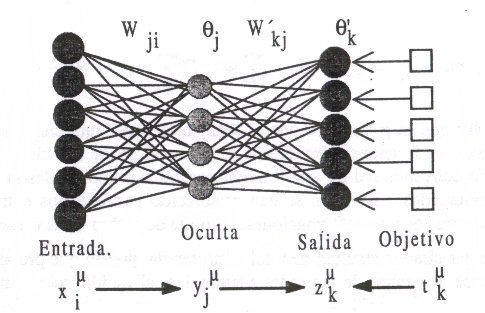
1. Calcular las derivadas parciales del error con respecto a los pesos \boldsymbol W^{(2)} que unen a la última capa oculta con la de salida.
2. Calcular las derivadas parciales del error con respecto a los pesos \boldsymbol W^{(1)} que unen la capa de entrada con la oculta.
3. Ajustar los pesos de cada neurona para reducir el error.
4. Repetir el proceso varias veces por cada par de entradas-salidas de prueba.

O = Xj\*F(netj)

# El perceptron multicapa (MLP)

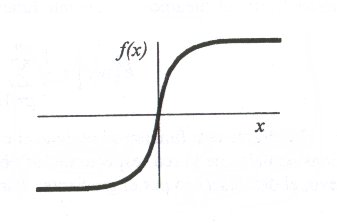
Este es uno de los tipos de redes más comunes. Se basa en otra red mas simple llamada perceptrón simple solo que el número de capas ocultas puede ser mayor o igual que una. Es una red unidireccional (feedforward). La arquitectura típica de esta red es la siguiente:

**Figura 3-2. Representación de un Perceptrón Multicapa (MLP)**



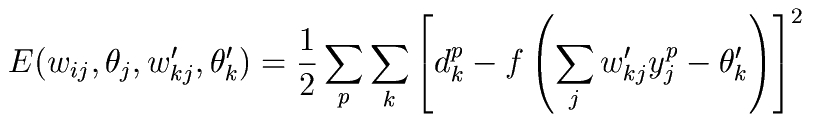
Las neuronas de la capa oculta usan como regla de propagación la suma ponderada de las entradas con los pesos sinápticos wij y sobre esa suma ponderada se aplica una función de transferencia de tipo sigmoide, que es acotada en respuesta.

**Figura 3-3. Forma funcional de una sigmoide**



El aprendizaje que se suele usar en este tipo de redes recibe el nombre de retropropagacion del error (backpropagation). Como funcion de coste global, se usa el error cuadratico medio. Es decir, que dado un par (**x**k, **d**k) correspondiente a la entrada k de los datos de entrenamiento y salida deseada asociada se calcula la cantidad:

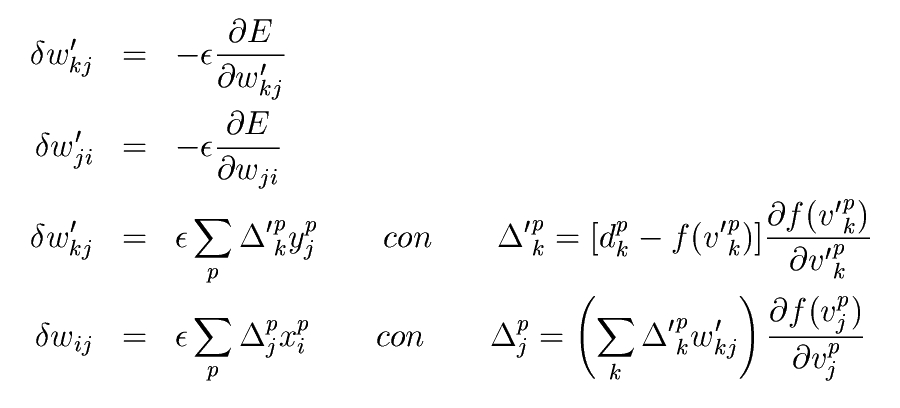
**Ecuación 3-1. Error cuadrático medio**



que vemos que es la suma de los errores parciales debido a cada patrón (índice p), resultantes de la diferencia entre la salida deseada dp y la salida que da la red f(.) ante el vector de entrada xk. Si estas salidas son muy diferentes de las salidas deseadas, el error cuadratico medio sera grande. f es la función de activación de las neuronas de la capa de salida e y la salida que proporcionan las neuronas de la ultima capa oculta.

Sobre esta función de coste global se aplica algun procedimiento de minimización. En el caso del MLP se hace mediante un descenso por gradiente. Las expresiones que resultan aplicando la regla de la cadena son las siguientes:

**Ecuación 3-2. Términos delta**



Siendo yk las salidas de la capa oculta.

El aprendizaje por backpropagation queda como sigue:

1. Inicializar los pesos y los umbrales iniciales de cada neurona. Hay varias posibilidades de inicialización siendo las mas comunes las que introducen valores aleatorios pequeños.
2. Para cada patrón del conjunto de los datos de entrenamiento
   1. Obtener la respuesta de la red ante ese patrón. Esta parte se consigue propagando la entrada hacia adelante, ya que este tipo de red es feedforward. Las salidas de una capa sirven como entrada a las neuronas de la capa siguiente, procesandolas de acuerdo a la regla de propagación y la función de activación correspondientes.
   2. Calcular los errores asociados según la ecuación 3-2
   3. Calcular los incrementos parciales (sumandos de los sumatorios). Estos incrementos dependen de los errores calculados en 2.b
3. Calcular el incremento total ,para todos los patrones, de los pesos y los umbrales según las expresiones en la ecuación 3-2
4. Actualizar pesos y umbrales
5. Calcular el error actual y volver al paso 2 si no es satisfactorio.

