ARQUITECTURA GENERAL DE UN PERCEPTRÓN MULTICAPA SUPERFICIAL

En la figura 1 presentamos la estructura del Perceptron Multicapa Superficial (MLP de sus siglas en inglés *Multi Layer Perceptron*), que a diferencia del Perceptron y del Adaline, posee al menos tres niveles de neuronas, el primero es el de entrada, luego viene un nivel o capa oculta y finalmente el nivel o capa de salida.

En las redes neuronales artificiales el término conectividad se refiere a la forma como una neurona de una capa cualquiera está interconectada con las neuronas de la capa previa y la siguiente. Para el MLP, la conectividad es total porque si tomamos una neurona del nivel de entrada, ésta estará conectada con todas las neuronas de la capa oculta siguiente, una neurona de la capa oculta tendrá conexión con todas las neuronas de la capa anterior y de la capa siguiente. Para la capa de salida, sus neuronas estarán conectadas con todas las neuronas de la capa oculta previa, para mayor claridad observemos la figura 1. Por lo general, se le implementan unidades de tendencia o umbral con el objetivo de hacer que la superficie de separación no se quede anclada en el origen del espacio *n*-dimensional en donde se esté realizando la clasificación.

La función de activación que utilizan las neuronas de una red MLP en la capa oculta suele ser tangente-sigmoidal, sigmoidal y recientemente tipo ReLU. Para la capa de salida se puede trabajar neuronas con función de activación sigmoidales, lineales o softmax. El tipo de función de activación que se trabaja en la salida depende del problema que se esté resolviendo que en general puede ser de dos tipos.

- De regresión
- De clasificación

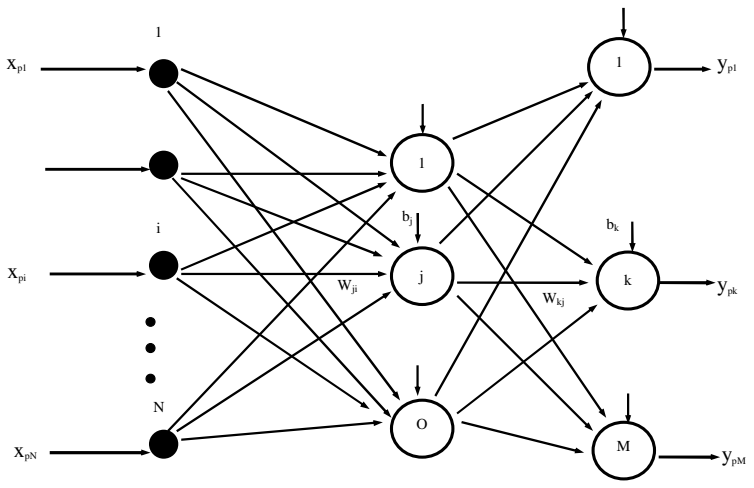

Fig. 1 Arquitectura General de un MLP

ENTRENAMIENTO DE UN MLP

Muchos autores traducen al español el nombre de este algoritmo, quizá la mejor acepción para el término *Backpropagation* es la de retropropagación o propagación inversa pues nos da una idea conceptual de la esencia del algoritmo, justamente de propagar el error de la capa de salida hacia las capas ocultas; sin embargo, el nombre anglosajón se ha aceptado en la mayoría de las publicaciones en español y lo usaremos en este libro.

Se sabe que uno de los principales inconvenientes que tiene el Perceptron es su incapacidad para separar regiones que no son linealmente separables y lo ilustramos al aplicarlo en la solución de un problema simple como la separación de las salidas de una función lógica XOR. Sin embargo, Rosenblatt ya intuía que un Perceptron multicapa si podía solucionar este problema pues, de esta manera, se podían obtener regiones de clasificación mucho más complejas. Sin embargo, persistían algunas interrogantes sin respuesta ¿Cómo entrenar un Perceptron multicapa? ¿Cómo evaluar el error en las capas ocultas si no hay un valor deseado conocido para las salidas de estas capas?

Una respuesta a estos interrogantes la planteó formalmente Werbos, cuando planteó el algoritmo de aprendizaje *Backpropagation*. Algoritmo que debe su amplia difusión y uso a David Rumelhart. Como el error de la capa de salida es el único que puede calcularse de forma exacta, el algoritmo propone propagar hacia atrás este error para estimar el error en las salidas de las neuronas de las capas ocultas, con el fin modificar los pesos sinápticos de estas neuronas.

Nomenclatura para la Redes Superficiales

Antes de formular matemáticamente el algoritmo, con la ayuda de la figura 1, definamos la notación que seguiremos a lo largo de este capítulo.

$\boldsymbol{x_p}$	Patrón o vector de entrada
x_{pi}	Entrada *i-ésima* del vector de entrada x_p
N	Dimensión del vector de entrada. Número de neuronas en la capa de entrada
P	Número de ejemplos, vectores de entrada y salidas diferentes.
O	Número de neuronas de la capa oculta
M	Número de neuronas de la capa de salida, dimensión del vector de salida
w_{ji}^{o}	Peso de interconexión entre la neurona *i-ésima* de la entrada y la *j-ésima* de la capa oculta.
b_j^{o}	Término de tendencia de la neurona *j-ésima* de la capa oculta.
$Neta_{pj}^{o}$	Entrada neta de la *j-ésima* neurona de la capa oculta

i_{pj}	Salida de la *j-ésima* neurona de la capa oculta
f_j^o	Función de activación de la *j-ésima* unidad oculta
w_{kj}^s	Peso de interconexión entre la *j-ésima* neurona de la capa oculta y la *k-ésima* neurona de la capa de salida.
b_k^s	Término de tendencia de la *k-ésima* neurona de la capa de salida.
$Neta_{pk}^s$	Entrada neta de la *k-ésima* neurona de la capa de salida.
y_{pk}	Salida de la *k-ésima* unidad de salida
f_k^s	Función de activación de la *k-ésima* unidad de salida
d_{pk}	Valor de salida deseado para la *k-ésima* neurona de la capa de salida.
e_p	Valor del error para el *p-ésimo* patrón de aprendizaje.
α	Taza o velocidad de aprendizaje
δ^o_{pk}	Término de error para la *k-ésima* neurona de la capa de salida.
δ^h_{pj}	Término de error para la *j-ésima* neurona de la capa oculta *h*.
$f'^{\,o}_j$	Derivada de la función de activación de la *j-ésima* neurona de la capa oculta.
$f'^{\,s}_k$	Derivada de la función de activación de la *k-ésima* neurona de la capa de salida.

Gradiente Descendente Estocástico y Gradiente Descendente

El gradiente descendente se aplica a redes neuronales para la actualización de pesos sin embargo, tal y como lo definimos en el capítulo anterior, para su cálculo se lo habitual es usar todos los patrones de entrenamiento o el lote completo de datos (*batch*). En caso de tener muchos datos para entrenar la red, el cálculo del gradiente se puede hacer computacionalmente dispendioso e inclusive prohibitivo.
Para evitar esto existe una alternativa que es calcular el gradiente en un subconjunto de los datos de entrenamiento. A este subconjunto de datos se le denomina minilote (*minibatch*). La selección de los datos usados para el cálculo del gradiente se hace de manera aleatoria. A esto se le ha denominado gradiente descendente estocástico (GDE).
Generalmente el GDE converge mucho más rápido en comparación con GD, sin embargo, no se suele llegar tan cerca al mínimo de la función de pérdida como lo hace el GD. En la mayoría de los casos el GDE queda oscilando alrededor del mínimo mencionado. Este comportamiento lo podemos observar en la figura 3.2. En ella verificamos que si usan todos los datos de entrenamiento (GD) la convergencia al mínimo de la función de costo o pérdida es mucho más definida y suave. En el caso de usar un solo dato de entrenamiento (GDE) la convergencia es más errática sin embargo, se alcanza un punto muy cercano al mínimo buscado.

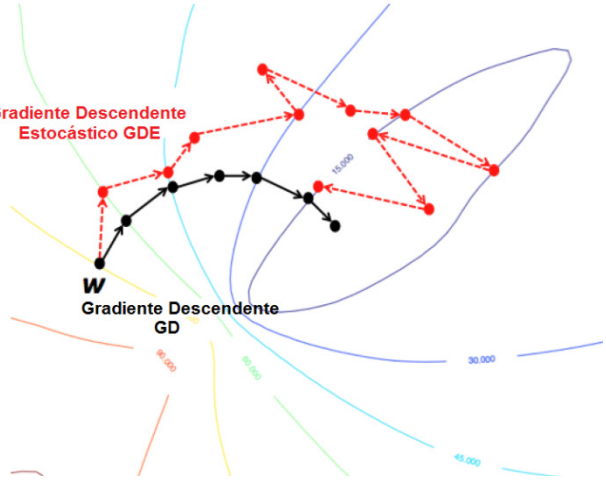

Fig. 3.2 Diferencia en el Comportamiento de GDE y GD
Fuente: https://wikidocs.net/3413

Aunque en uso del GDE se ha hecho muy popular principalmente en las redes profundas, usarlo en las redes superficiales es una buena estrategia para disminuir la carga computacional en el entrenamiento de las mismas.