Neurocomputación

Práctica 1:

Redes de McCulloh-Pitch. Perceptrón y Adaline.

Daniel Gutiérrez Navío
Matías Bender
Grupo 2461

3 de marzo de 2014

Índice

Índice de figuras

Introducción

Esta práctica, la primera de la asignatura de Neurocomputación, se tratará de conseguir una aproximación a redes neuronales, mediante la codificación de tres de las primeras redes neuronales: perceptrón simple, adaline, y red de McCulloh-Pitts, para la resolución de los problemas propuestos.

Diseño

Para la resolución de esta práctica, y de las venideras, hemos decidido realizar el desarrollo de una interfaz para redes neuronales en C++, para simplificar futuros trabajos. La interfaz principal, está compuesta de estructuras de neuronas y links. Una neurona es una simple unidad, con entrada y salida de datos, y un proceso adaptable para la transformación de una en otra.

El link, es un enlace pesado entre dos neuronas, que lleva la salida, multiplicada por su peso, a la entrada del objetivo, donde suma a lo que ya haya.

Mediante estos dos elementos, podemos construir cualquier red neuronal. Además, se ha diseñado una clase, perceptrón, con una estructura genérica adaptable, que puede usarse para la construcción de otras redes.

1. Transformada rápida de Fourier con recursión

En este apartado de la práctica, se deberá realizar una red de McCulloh-Pitts para la resolución del problema propuesto. Se utilizará la siguiente arquitectura de McCulloh-Pitts para realizar la red:

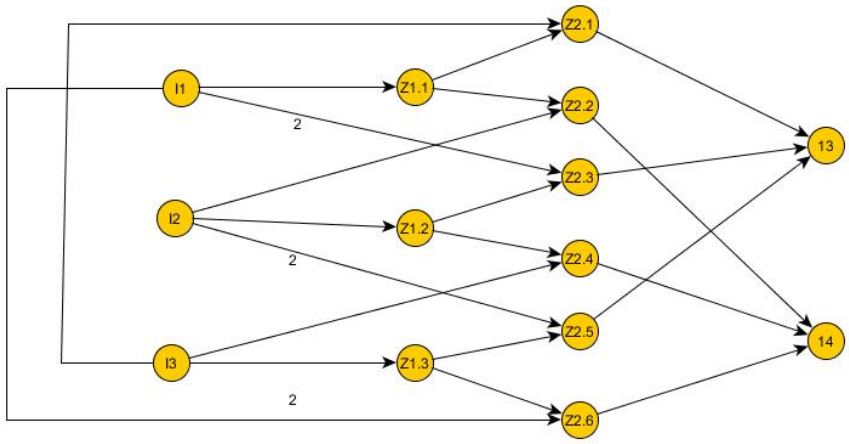

Figura 1: Red de McCulloh-Pitch

Usando como umbral en todas las neuronas 2, y sin usar ninguna sinapsis inhibitoria.

La implementación de esta red, como puede verse, se ha hecho de manera totalmente lineal, creando una estructura en capas, en las que no existe ningún tipo de retroalimentación, sino que una capa inferior proporciona salidas conectadas a las capas superiores (no a la inmediatamente superior necesariamente).

En nuestro caso, usamos 4 capas: una capa de entrada, una capa de salida, y 2 niveles de capas ocultas, provocando así un retraso de 3 épocas entre una entrada, y su correspondiente salida.

En caso de no haber entrada dado que no se ha cargado aún, el sistema devolverá 0 0 0, la salida neutra, que no proporciona ningún tipo de información.

Se han diseñado distintos ejemplos, y sus salidas, que pueden verse en el makefile escribiendo `makefile help-mcculloh`.

2. Perceptrón y Adaline

Introducción

En esta parte, se deberá realizar la implementación de dos algoritmos nuevos: un Perceptrón de capa simple, y un Adaline. Ambos redes neuronales simples para problemas lineales.

En ambos casos, son algoritmos de aprendizaje supervisado, que requieren de un tiempo de preparación, llamado entrenamiento, para su correcto funcionamiento.

2.1. Perceptrón

2.1.1. Introducción y diseño

El objetivo de este ejercicio consiste en construir un perceptrón simple, o monocapa, con una sola neurona de salida, para reconocer patrones. Usaremos la estructura de red neuronal que hemos descrito al principio de este documento, pero adaptandola al sistema de entrenamiento y clasificación de un perceptrón.

Este sistema consiste en aplicar una variación de la regla de hebb, es decir, en caso de error en clasificar un elemento de entrenamiento, aplicaremos una actualización de pesos. La actualización de pesos que aplicaremos será la siguiente: $w_{t+1} = w_t + \alpha * t * x$

Donde w, es el peso de la sinapsis en un momento concreto, α es la constante de entrenamiento ($0 < \alpha \geq 1$), y t es un valor de verdad, que corresponde a la clase que es de verdad (1 si es clase 1, -1 si es clase 2, al ser bipolar).

Decidimos usar de condición de parada, un 100 % de acierto, es decir, una etapa completa sin un solo error. Pero como muy a menudo, al poder clasificar solo problemas lineares, esto no es posible, hemos decidido una cantidad de épocas de entrenamiento. Usaremos una cantidad de épocas elevada, 100, aunque luego comentaremos esta cantidad y miraremos los problemas acotados en caso de que se detenga el proceso de aprendizaje.

2.1.2. Descripción de los resultados

Tras preparar el perceptrón de la manera que se ha explicado, ejecutamos el programa para obtener datos estadísticos de el proceso de aprendizaje.

El resultado, es de un 93 % de acierto en entrenamiento, que cambia a 95 % de acierto en el testing.

El resultado, como podemos comprobar, es realmente muy eficaz, dado que podemos ver que se adapta de una manera muy buena a el problema propuesto.

En la siguiente gráfica, podemos apreciar la evolución durante el entrenamiento (acotado).

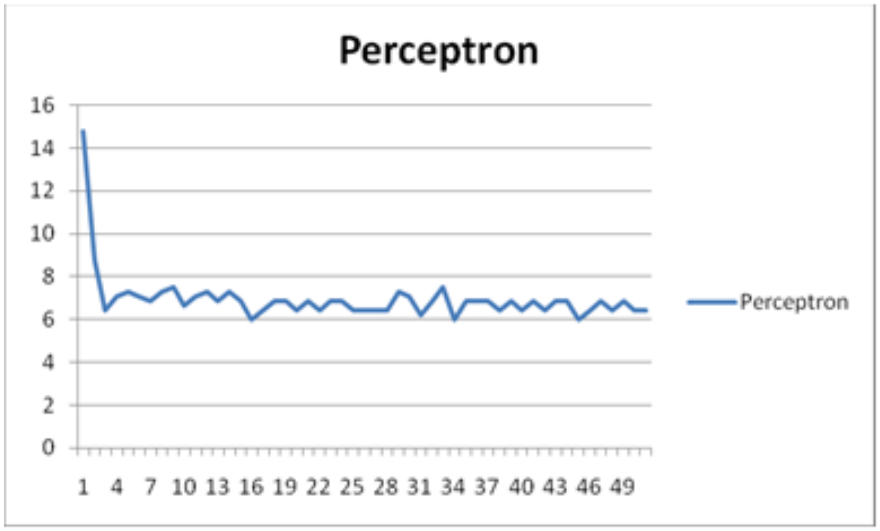

Figura 2: Evolución del entrenamiento del perceptrón

Podemos comprobar, que se mantiene constante gran parte del tiempo el error, por lo que podemos determinar que, usando un perceptrón, no podemos superar cierta cota en este problema de error, suponiendolo a ruido o casos no lineales.

2.2. Adaline

2.2.1. Introducción y diseño

En este ejercicio, se nos pide modificar la red creada en el anterior, el perceptrón, para crear un Adaline. El adaline es una red neuronal, basada también en aprendizaje de Hebb, pero con graves modificaciones, para orientarse a la reducción del error cuadrático medio. El error cuadrático medio (ecm), es una medida usada para el error promedio en estimación. Para calcularlo, se tiene que encontrar el valor esperado del cuadrado de la diferencia entre <X>(el estimador), y X, aplicando la fórmula:

$ecm = \frac{sqrt(sum(xi-<X>)^2)}{n}$

siendo <X>, $(\frac{sum(x_i)}{n})$, podemos sustituir, dándonos la fórmula

$ecm = \frac{sqrt(sum(xi-(\frac{sum(x_i)}{n}))^2}{n}$

La regla que reduce este error, se obtiene mediante su gradiente, dándo a usar la conocida como llamada regla delta para la actualización de pesos:

$w_{i+1} = w_i + \alpha * (t - y) * x$

Esta regla, a diferencia del perceptrón, que solo actualiza en caso de error, se aplica siempre.

2.2.2. Descripción de los resultados

Para exponer los resultados, usaremos la misma estructura que empleamos en el caso del perceptrón, solo que además añadiremos una gráfica exponiendo la variación del error cuadrático medio, para observar la eficacia de este algoritmo (cosa que no se hizo en el perceptrón, al no ser considerado relevante para el problema, ya que el perceptrón ignora (aunque afecta indirectamente) el error cuadrático medio, y no es motivo de comparación).

El resultado del entrenamiento, con un máximo de épocas acotado, es el siguiente, viéndolo comparativamente el ECM con el error:

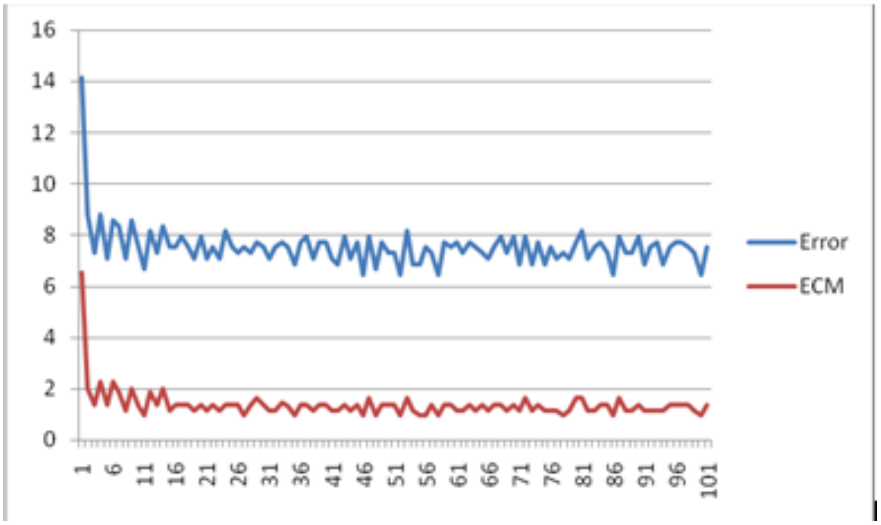

Figura 3: Adaline: ECM vs Error

El error al final del entrenamiento, como ha podido apreciarse, es de XXX %, mientras que en el test, es de XXX %. Esto, como se mencionó, varía con cada ejecución al hacerse una distribución aelatoria de los datos.

2.3. Parámetros de aprendizaje

Para realizar este ejercicio, mostraremos una tabla con los resultados de variar, con un máximo de 100 épocas, la constante de aprendizaje entre 0,1 y 1, tanto en adaline como en pereceptrón.

Como se puede ver, al no llegar a converger nunca, no se ha podido comprobar exactamente el efecto. Sin embargo, se ha podido apreciar que tarda más en estabilizarse cuanto menor es la constante de aprendizaje (la diferencia sin embargo, no es demasiado considerable en este intervalo).

Constante	Adaline	Perceptron
0.2	91.32%	92.86%
0.3	91.72%	91.32%
0.5	86.24%	92.12%
0.6	92.32%	93.16%
0.7	91.86%	92.93%
0.9	92.47%	92.89%

Figura 4: Constantes de aprendizaje

2.4. Entrenamiento-test completo

La variación se ha realizado, haciendo que pueda recibir el main 2 argumentos, un primer argumento que sea el conjunto de entrenamiento, y otro segundo argumento, el de test. En caso de ser de conjuntos distintos, el programa no se hace responsable y DARÁ MUY PROBABLEMENTE UN ERROR.

A continuación se muestra el resultado de analizar los archivos propuestos, con número de épocas hasta la convergencia, en caso de existir (límite: 100 epocas).

NAND

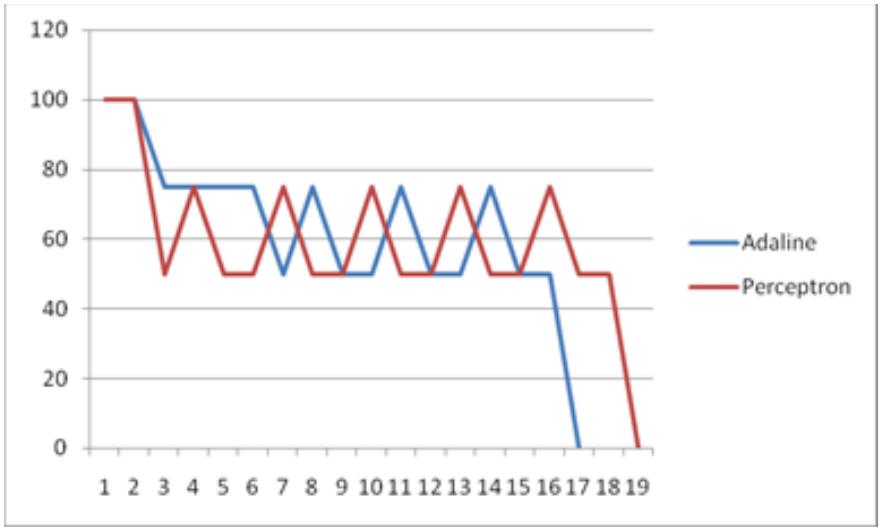

Figura 5: NAND

NOR

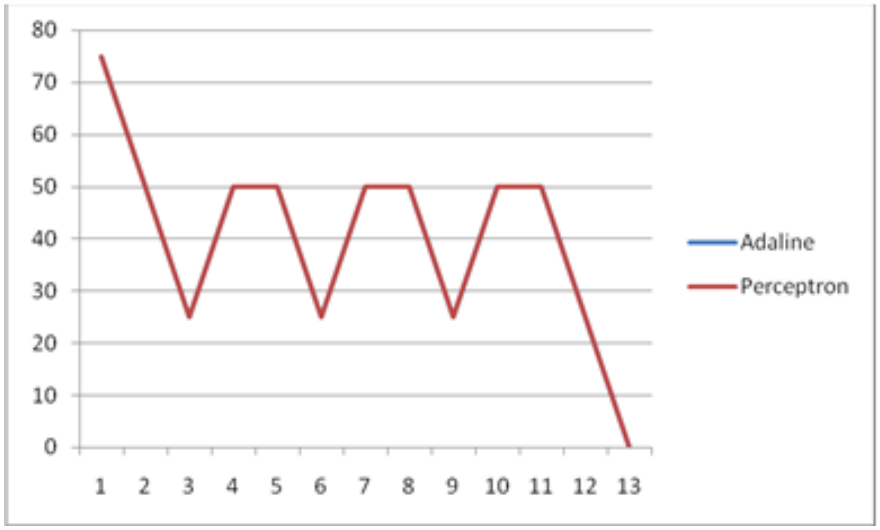

Figura 6: NOR

Nótese que aquí ambas coinciden

XOR

Podemos observar que aquí siempre da error. Podría dar al principio algún caso de éxito, dependiendo de los valores iniciales.

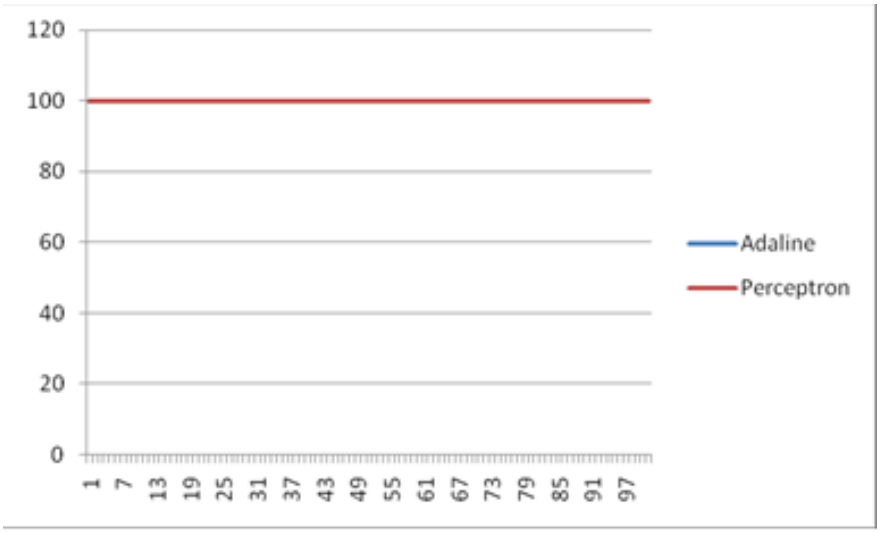

Figura 7: XOR

Como podemos comprobar, xor no llega a ninguna convergencia. Esto es debido a que xor no es linealmente separable, mientras que los demás al ser linealmente separables, si que pueden ser resueltos completamente.

2.5. Clasificación de datos desconocidos

Se ha aplicado el problema propuesto tanto al perceptrón como al adaline. Se muestran a continuación las gráficas de entrenamiento (acotadas a 100 epocas), de ambos, en el caso de usar con una división 2/3 training, 1/3 test, en el archivo problema_real2.txt.

La gráfica de la evolución de el entrenamiento en perceptrón y adaline es la siguiente:

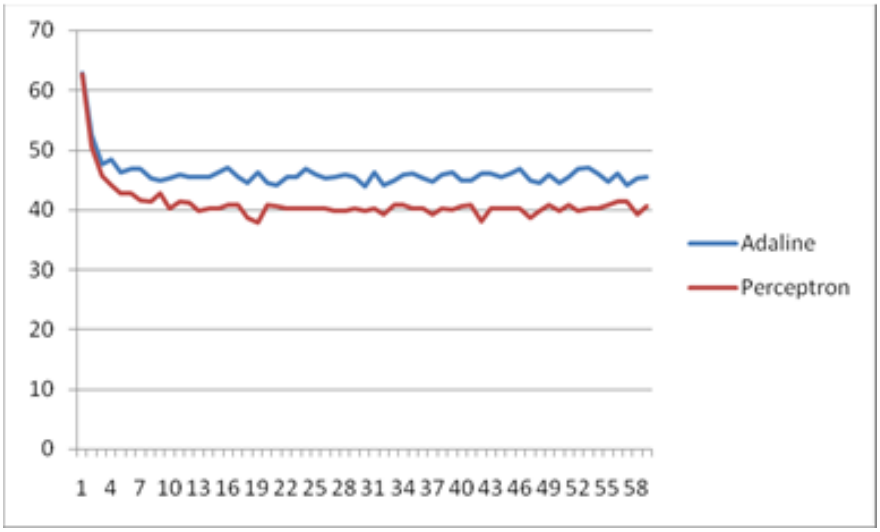

Figura 8: Evolución del entrenamiento del Perceptrón y Adaline

En ella podemos ver que el perceptrón tiene un índice de error menor que el adaline. Eso es debido a que el perceptrón funciona pensando en el error final, mientras que el adaline en el error cuadrático medio, siéndo este visiblemente más suave.

Por último, se muestra a continuación las gráficas de entrenamiento de usar el fichero entero como entrenamiento. Los resultados de el problema, irán en un archivo adjunto a esta memoria.

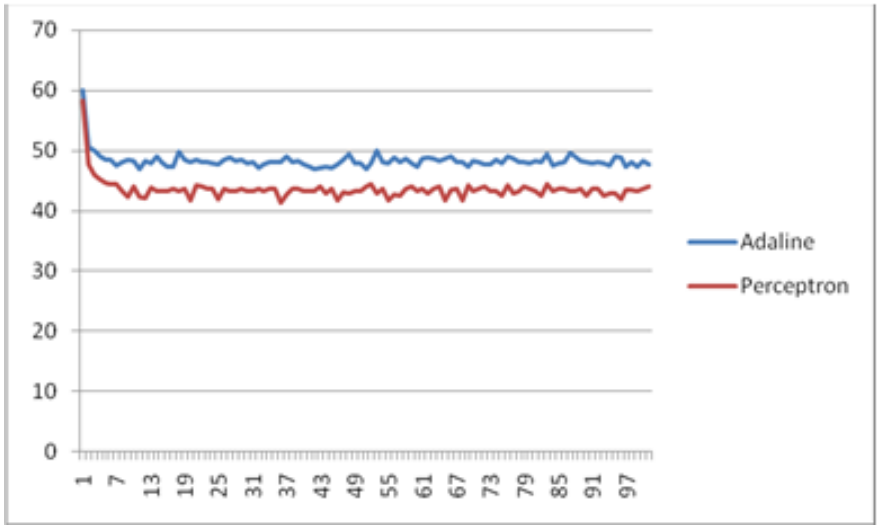

Figura 9: Evolución del entrenamiento 'completo' del Perceptrón y Adaline

En ella se puede ver un desarrollo similar a la anterior, en la que el perceptrón muestra, durante el entrenamiento, un mejor comportamiento que el adaline, al tener casi un 10 % de acierto más que el adaline.

2.6. Añadir parámetros no lineales

Este ejercicio no es propiamente un ejercicio, ya que basta con añadir unos cuantos datos aleatorios, que sean no lineales (ruido). Para ello, hemos introducido en los datos ciertos datos que nos hemos inventado, y hemos observado que la precision empeora, pero al no converger ya de por sí, el impacto es menor del que esperamos.

Conclusiones

En esta práctica hemos aprendido a implementar redes neuronales para la resolución de sistemas lineales (o problemas lineales). Pese a la utilidad de ésto, teóricamente, en la realidad, como se ha podido comprobar durante la práctica, los problemas lineales no son demasiado comunes, por lo que un sistema imperfecto como éste no tiene demasiada utilidad en un caso real (ruido, así como complejidad de los problemas). Sin embargo, estos algoritmos luego servirán de base para otros más potentes con capacidad de división no lineal.

Además, en esta práctica hemos creado, en el lenguaje de programación `C++`, un "framework"de redes neuronales, para realizar una arquitectura de redes neuronales genéricas, que nos servirá en las siguientes prácticas.