Net-Mind

¿Sueñan las redes artificiales con vacas mecánicas?

Andrea Cimmino Arriaga

Alumno de Inteligencia Artificial 2, Ingeniería Informática
Universidad de Sevilla
banse27@gmail.com

Abstract. El objetivo principal del presente trabajo era la implementación de un generador de distintas redes neuronales, con diferentes topologías, que además podrían ser entrenadas con un algoritmo de retropropagación, momentum y validación cruzada usando la parada temprana.

También se han realizado distintos experimentos con varios ficheros de *benchmarks,* extraídos de [1] para ver la eficiencia de los redes implementadas sobre un campo real.

Los resultados obtenidos pueden observarse en la última sección de este mismo trabajo y, como se puede ver, son discretamente buenos.

1 Introducción

El presente trabajo se basa en dos pilares fundamentales, las redes neuronales artificiales (RNA) y un conjunto de *benchmarks,* pertenecientes a PROBEN1 [1].

Los principales **objetivos de este trabajo** se pueden dividir de la siguiente manera:

- Experimentalmente hay que **implementar un generador de redes neuronales**, al que se le puede indicar una topología especifica para la red, y que debe ser capaz de entrenarla. Podemos distinguir distintos subobjetivos:

    1. Implementar un generador de RNAs.
    2. Un algoritmo de entrenamiento.
    3. Realizar una serie de experimentos.
    4. Analizar el rendimiento y los resultados.

- Teóricamente, ademas de comprender los algoritmos a implementar, tenemos **la exposición de los resultados obtenidos y la investigación que conlleva del presente artículo**, necesaria para escribirlo. Podemos resumirlo todo en los siguientes subobjetivos:

    1. Comprender conceptualmente las redes neuronales para poder trabajar con ellas y luego explicarlas.
    2. Comprender los distintos algoritmos a implementar.
    3. Investigar que medidas son las mejores para analizar los resultados obtenidos.

La estructura de este documento es la siguiente: En el apartado 2, **Preliminares**, hago una pequeña, pero exhaustiva, introducción a las redes neuronales artificiales. En la sección 2.1 se explica cómo están formadas las distintas topologías que pueden adoptar, cómo funcionan, algunas aplicaciones y, por último, comento las distintas ventajas y desventajas que poseen. Posteriormente en el apartado 2.2 describo en qué consiste PROBEN1 [1], porqué se reúnen una serie de pruebas, tanto para entrenar, como para *testear*, las redes neuronales artificiales; así como porqué es necesaria una estandarización de los *benchmarks*.

El siguiente apartado, el 3, contiene información teórica sobre los algoritmos presentes en [4], o mejoras de estos que también se explican en [4]. Son los que se han implementados en el trabajo práctico: retropropagación (sección 3.1), momentum (sección 3.2), validación cruzada (3.3) y parada temprana (sección 3.4).

A continuación, en el apartado 4, expondré de forma teórica los distintos experimentos que llevaré a cabo. En la sección 4.1, hablo de ellos de forma teórica y, en la 4.2, los resultados obtenidos, tras entrenar la red y lanzar los datos asociados a cada experimento.

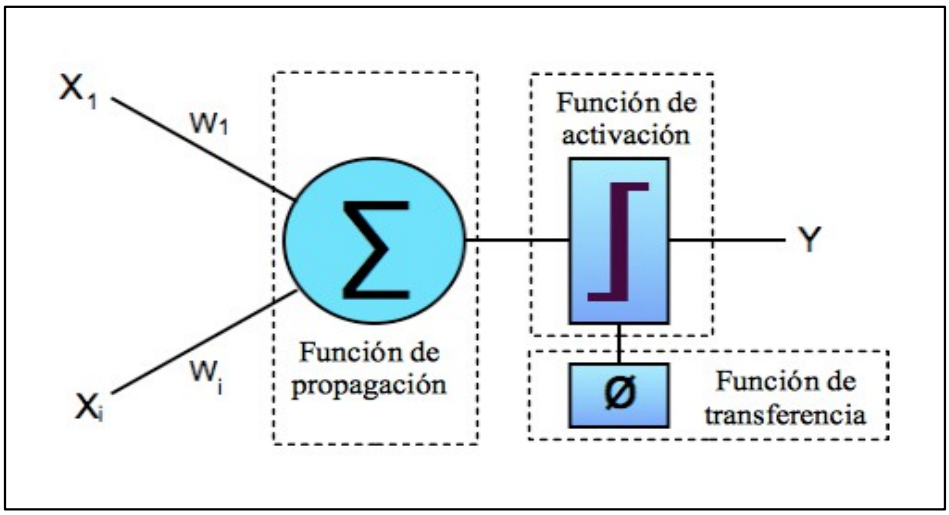
Por último, en el apartado 5, hablo sobre las conclusiones que podemos extraer, tanto de los resultados obtenidos en el apartado de experimentos, como de otras a las que he llegado durante la implementación y la experimentación.

2 Preliminares.

2.1 Redes Neuronales Artificiales.

Con el fin de comprender perfectamente el trabajo, en esta sección, explicaré detalladamente lo **que es una red neuronal**.

Como la propia expresión indica, una **red neuronal es una red de neuronas**, así que **el elemento básico son precisamente éstas**. Las artificiales, que se basan en las naturales, **tienen una estructura** "conceptual"[1] **como la que sigue**:



- **X1..Xi son las entradas** de la red que, a su vez, tienen asociados unos pesos: **W1..Wi. Estas aristas se denominan sinapsis[2].**
- **Función de propagación o de excitación**: Consiste en el sumatorio de cada entrada multiplicada por su peso asociado. El resultado pasa a la siguiente función.
- **Función de activación**: Modifica el valor obtenido de la anterior función, puede no existir (como en el caso de la implementación de este trabajo).
- **Función de transferencia**: Se utiliza para acotar la salida de la neurona, algunas de las mas conocidas son la sigmoidea (usada en el trabajo) y la umbral.

Una vez visto lo anterior, podemos resumir la neurona como una caja negra que recibe unas entradas con unos pesos asociados y devuelve un valor de salida.

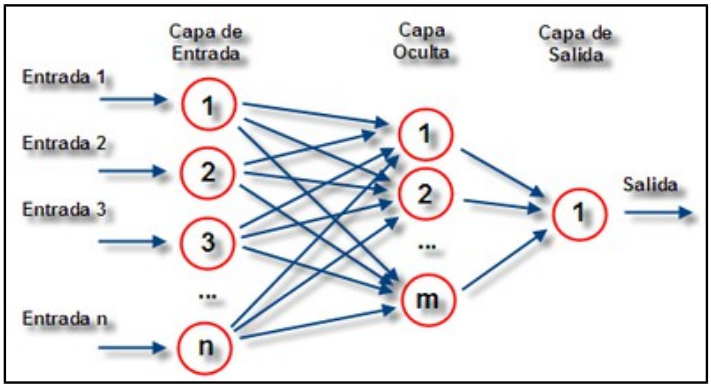
Ahora bien, esas entradas y esas salidas, ¿Qué son ? Para responder a esta pregunta es necesario retomar una perspectiva más general. Recordemos que una red neuronal

[1] "Conceptual", porque me refiero a la teoría y no a la implementación informática.

[2] En varios documentos se recomienda evitar esta terminología para las redes artificiales, aún así aportan mucha claridad conceptual a la hora de explicarlas, motivo por el que la uso.

dijimos que era precisamente eso; muchas neuronas interconectadas. Siendo así esas entradas esta claro que serán las salidas de otras neuronas y las salidas de la presente neurona sera la entrada de otras.

Podemos ir haciéndonos una idea de la estructura de una red, veamos una genérica:



Como vemos en la ilustración superior, **la red se compone de una serie de capas que, a su vez, poseen un número variable de neuronas. La red posee unas entrada,** que son sinapsis que entran directamente a las neuronas con un peso unitario, **y unas salidas**, que no poseen pesos, como es de esperar.

Podemos dividir la red en tres partes:

- Capa de Entrada: Es obligatoria en cualquier red, recibe los datos de entrada y los propaga a la siguiente capa. Las entradas tienen un peso unitario asociado.
- Capa oculta: Esta capa puede también no existir, además no tiene por que ser sólo una. En cada capa oculta no tiene porqué haber el mismo número de neuronas.
- Capa de salida: Obligatoria en cualquier red, simplemente devuelve el valor obtenido para una entrada dada.

La topología de las redes puede ser muy variada, pero también podemos clasificarla en tres tipos:

- Una capa de entrada y otra de salida.
- Una capa de entrada, otra de salida y las capas ocultas tienen todas el mismo número de neuronas.
- Una capa de entrada, otra de salida y cada capa oculta tiene un número distinto de neuronas.

Para terminar hay que hablar de un elemento que aún no hemos visto. Existe una neurona denominada **neurona umbral**. Dicha neurona puede estar conectada con todas las neuronas o sólo con algunas. En el caso de mi red, está conecta a todas las neuronas menos a las de la capa de entrada. **La neurona umbral no recibe ningún valor de entrada y siempre devuelve como salida un valor constante** (-1 en mi trabajo), lo que si varía es el peso asociado a cada sinapsis.

Por lo tanto, **¿Cómo funciona la red?** Se le introducen unos valores en las sinapsis de entrada y esa información se procesa en cada neurona de la primera capa. A continuación, se propaga a la siguiente, y así, hasta llegar a la capa de salida, donde en lugar de propagarse se obtiene un resultado (una estimación de un valor esperado o una inferencia sobre un posible valor).

Evidentemente nada más crear la red, si introducimos un valor, sabiendo previamente lo que queremos obtener, la salida de la red no se aproximará nada al valor esperado. **Para que la red neuronal aproxime matemáticamente algún proceso**, o pueda tomar alguna decisión en base a unos parámetros, **es necesario entrenar la red**. **¿En qué consiste esta fase?** Como vimos antes en la red, realmente lo único que podría irse variando son los pesos de las sinapsis. Precisamente eso es lo que hace el proceso de entrenamiento. Consiste, mediante ejemplos que sabemos de antemano lo que deben devolver, procesar los datos y obtener una salida. En base al error producido se reajustan los pesos.

Existen distintos algoritmo de entrenamiento, pero más o menos todos funcionan de una forma parecida. De este tema se hablará más en el apartado 3.1 del trabajo.

Las RNA se suelen aplicar en problemas como reconocimiento de voz, imágenes, etc. o para problemas de patrones, como pueden ser el campo financiero o el de la climatología, entre otros muchos.

Las ventajas de las redes neuronales artificiales pueden resumirse en los siguientes puntos:

- **Aprendizaje**: Se lleva a cabo en una etapa controlada en la que sabemos y podemos calcular la eficiencia de la red, lo que permite entrenarla y saber que cuando la vayamos a usar tenemos una medida empírica de su calidad.
- **Auto-organización**: La red al crear su propia representación de la información descarga al usuario de eso.
- **Flexibilidad**: Las RNA pueden manejar entradas que posean ruido o interferencias sin ningún problema consiguiendo de todas formas los resultados esperados.
- **Tolerancia a fallos**: Debido a que cada neurona almacena su información, aunque parte de la red falle, puede seguir dando buenos resultados.
- **Operaciones en tiempo real**: además de la relativa baja complejidad computacional hoy día es posible paralelizar (mediante hardware y software), las operaciones que realizan las RNA, obteniendo respuesta muy rápidas.

Por otro lado las RNA también poseen desventajas:

- El mayor problema es que el grado de paralelismo, aunque está entre las ventajas, es relativamente pequeño. Si pudiéramos darle mayor paralelismo a las redes, éstas serían mucho más eficientes.
- Otro problema es el tamaño: no somos capaces de reproducir el número de neuronas del cerebro humano. Aunque se consiguen grandes redes, aún no son lo suficientemente grandes para operaciones excesivamente complejas.
- Sobreajuste: aunque hay algoritmos para evitar el sobreajuste, sigue siendo un problema muy presente.
- Mínimos Locales: la red, si no se toman precauciones, puede atascarse en mínimos locales.

2.2 PROBEN1

PROBEN1 es uno de los pilares en los que se basa entre trabajo, **consiste en un conjunto de problemas para el aprendizaje de las redes neuronales y una serie de reglas para la estandarización los *benchmarks.*** Sobre el porqué un conjunto de *benchmarks,* el propio autor del documento [1] nos contesta que el motivo de escribir PROBEN1 fue debido a que la investigación sobre el aprendizaje de las redes es muy pobre hoy día. En el documento se presentan una serie de motivos por los que los investigadores justifican esta situación y, a continuación, los rebate uno a uno, demostrando que realmente el panorama actual de este campo no está poco desarrollado por limitaciones tecnológicas, sino más bien humanas y de organización.

Los aspectos del aprendizaje, que pueden estudiarse con los problemas de PROBEN1, son la velocidad de aprendizaje, la capacidad de generalización de la red, la facilidad y el impacto de la variación de parámetros por parte del usuario.

Por otro lado dijimos que PROBEN1 también consistía en una serie de reglas para la estandarización de *benchmarks*. El motivo de éstas, según el propio autor, es para dar soporte a la ciencia y la investigación, de tal manera que sea posible incrementar la calidad de las evaluaciones, usando redes. Sólo si los resultados de los distintos problemas son comparables y reproducibles, se habrá producido una mejora real en el campo de la investigación. Es importante también tener un estándar para los ficheros de entrada, la configuración inicial, las medidas tomadas y la documentación de dichos resultados. Así que para poder estudiar el rendimiento real y un comportamiento real de las redes y, con el fin de dar un soporte de rigor científico a las pruebas, se crean estas reglas de estandarización.

Según el autor de PROBEN1, los principio para la formulación de las reglas de *benchmarking* son las siguientes:

- **Validez**: En el sentido que los resultados obtenidos no sean debidos a de factores aleatorios, sino a un cálculo científico. Aunque se sigan las reglas que se detallan en algunos apartados del documento, no se puede asegurar que siempre se pueda conseguir este punto.

- **Roproducibilidad**: Esta regla exige la especificación de la configuración inicial de la red con el fin de que otros investigadores puedan reproducir tal cual el experimento.
- **Comparabilidad**: Es obligatorio poder comparar para un mismo problema inicial los resultados obtenidos de forma instantánea.

Por otro lado, se nos presenta también una estructura para los ficheros de pruebas. Teniendo un fichero con los datos, es importante elegir qué datos se usan para el entrenamiento y cuáles para el *testing*. De hecho esta elección puede llevar a entrenar de una mejor o peor manera la red. Por este motivo en el artículo [1] se nos presenta la siguiente estructura: los datos se dividirán en dos grandes conjuntos, el conjunto de aprendizaje y el de *testing*. El primero se encargará de medir el rendimiento, mientras que el segundo de entrenar a la red. A su vez el conjunto de aprendizaje se divide en dos subconjuntos más, el de entrenamiento y el de validación. El primero es usado para el aprendizaje de la red, el segundo para comprobar que el aprendizaje se está llevando a cabo de forma correcta, medir el error de estimación y tener un criterio para saber cuando la red esta lista y puede dejar de entrenarse.

Por lo tanto en los ficheros debe especificarse qué ejemplos pertenecen a cada conjunto de los anteriores, el número de entradas que tiene la red, y el número de salidas.

En el artículo se nos habla acerca del algoritmo de aprendizaje y se hace especial hincapié que el usado no importa, lo importante es especificar todos y cada uno de los parámetros que utilicé, además, indicar los parámetros seleccionables y modificables y especificar la sensibilidad que tiene la red ante posibles variaciones de los mismos.

Acerca de las medidas de error a usar también se hace mención en la sección 2.6 de este mismo trabajo.

Por último, también se hace mención al tipo de redes que se deberían usar (multicapa), la importancia de la inicialización aleatoria (que según los valores que se asignen a las sinapsis la red puede converger mas rápidamente) y, por último, sobre los resultados del entrenamiento se explica que los parámetros que más nos interesa medir son la medida de generalización, que es el error del conjunto de *testing*, y luego de forma menos importante, otras como los “epochs”, el número de ejemplos procesados, etc.

Para terminar, la referencia que se hace a la estandarización sobre la exposición de los problemas es la que sigue. Para cada problema se deberían de especificar la siguiente información:

1. Problema: nombre, versión/variante.
2. Conjunto de entrenamiento, de validación y de *testing*.
3. Red: nodos, conexiones y funciones de activación.
4. Inicialización.
5. Parámetros del algoritmo.
6. Criterios de finalización.

7. Función con la que se midió el error y la normalización de la misma en los resultados obtenidos.
8. Numero de ejecuciones.

Como vemos [1], es un valioso documento que pretende traer mayor rigor científico al campo de las redes neuronales, para ello propone todo lo anterior.

3 Algoritmos.

Para la realización del presente trabajo se han implementado los siguientes algoritmos:

- Retropropagación: Es el encargado de entrenar la red Neuronal.
- *Momentum*: Es una modificación que se realiza sobre el anterior algoritmo para evitar quedar atascados en mínimos locales.
- Validación Cruzada: descrito en el *papper* [1], se encarga de subdividir los datos de entrada de la manera más eficiente para poder entrenar la red con el primer algoritmo.
- Parada Temprana: es un método implementado en el algoritmo anterior que evita que se produzca el sobreajuste.

3.1 Algoritmo de Retroporpagación.

El algoritmo de retropropagación es un algoritmo de entrenamiento basado en otro denominado el método del descenso del gradiente.

Lo que hace este algoritmo es modificar los pesos de las sinapsis. Para ello utilizan un conjunto de ejemplos que poseen unas entradas y las salidas esperadas. Se introducen los datos en de entrada en la red y se obtiene la salida de la misma. Después se compara el resultado obtenido con el esperado, viendo el error producido, y en base a eso se retrocede, desde las neuronas de la capa de salida hasta las de entrada, corrigiendo los pesos asociados en base al error de las neuronas de la capa delantera.

Para resumir lo anterior, se calcula la salida obtenida de la red y en base al error con la esperada, se corrigen los pesos de las sinapsis, propagando el error por la red desde la salida hasta la entrada.

Matemáticamente el error en los nodos i-ésimos de la capa de salida se calculan con la siguiente formula

$$\Delta_i \leftarrow g'(in_i)(y_i - a_i)$$

- **g'(x)**: Representa la función de propagación.
- **ini**: Son las entradas la neurona multiplicadas por sus pesos.
- **ai**: es la salida obtenida en el nodo correspondiente de la red.
- **yi** : es la salida esperada.

A continuación para cada nodo j-ésimo perteneciente a la capa anterior a la de salida se realiza la siguiente operación. Iterativamente se repite el proceso hacia atrás tomando como nodo i-ésimo la capa delante de actual, que sera la capa j.

$$\Delta_j \leftarrow g'(in_j) \sum_i w_{ji} \Delta_i$$

- Wji representa la arista que une el nodo j-ésimo con el de la capa de enfrente que es el nodo i-ésimo.

Conforme calculamos el error en cada nodo, actualizamos también los pesos de salida de este (las sinapsis que lo unen a la capa siguiente, la i-ésima) usando la siguiente fórmula:

$$w_{ji} \leftarrow w_{ji} + \eta a_j \Delta_i$$

- La letra que esta junto a la aj (la salida de la neurona j) se denomina constante de aprendizaje y es una constante.

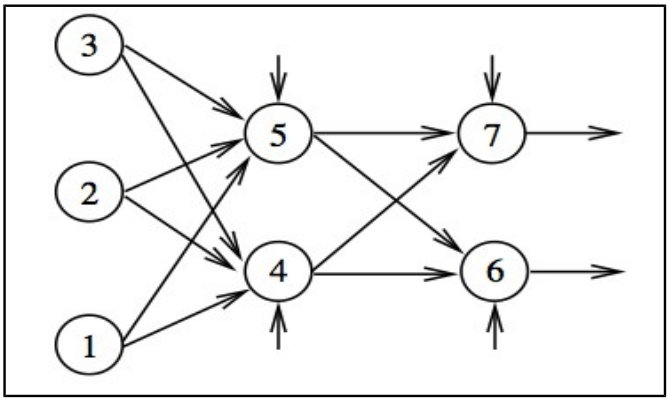
Como vemos para corregir el error de una arista se tiene en cuenta el error que se ha producido en el nodo que tiene delante.

Cuando se lleva a cabo este proceso hay que tener en cuenta la neurona umbral, que también posee peso umbral.

Por último si se esta usando la función sigmoidea, las operaciones anteriores pueden simplificarse de la siguiente manera:

- Para la capa de salida → $\Delta_i \leftarrow a_i(1 - a_i)(y_i - a_i)$
- Para el resto de capas → $\Delta_j \leftarrow a_j(1 - a_j)\sum_i w_{ji}\Delta_i$

Para entender bien el algoritmo, veamos un ejemplo. Imaginemos la siguiente red:



Los cálculos asociados a cada neurona serán:

Capa	Unidad	Cálculos que se realizan
Salida	7	$\Delta_7 = a_7(1-a_7)(y_7-a_7)$
		$w_{0,7} \longleftarrow w_{0,7} + \eta a_0 \Delta_7$
	6	$\Delta_6 = a_6(1-a_6)(y_6-a_6)$
		$w_{0,6} \longleftarrow w_{0,6} + \eta a_0 \Delta_6$
Oculta	5	$\Delta_5 = a_5(1-a_5)[w_{5,6}\Delta_6 + w_{5,7}\Delta_7]$
		$w_{0,5} \longleftarrow w_{0,5} + \eta a_0 \Delta_5$
		$w_{5,6} \longleftarrow w_{5,6} + \eta a_5 \Delta_6$
		$w_{5,7} \longleftarrow w_{5,7} + \eta a_5 \Delta_7$
	4	$\Delta_4 = a_4(1-a_4)[w_{4,6}\Delta_6 + w_{4,7}\Delta_7]$
		$w_{0,4} \longleftarrow w_{0,4} + \eta a_0 \Delta_4$
		$w_{4,6} \longleftarrow w_{4,6} + \eta a_4 \Delta_6$
		$w_{4,7} \longleftarrow w_{4,7} + \eta a_4 \Delta_7$
Entrada	3	$w_{3,4} \longleftarrow w_{3,4} + \eta a_3 \Delta_4$
		$w_{3,5} \longleftarrow w_{3,5} + \eta a_3 \Delta_5$
	2	$w_{2,4} \longleftarrow w_{2,4} + \eta a_2 \Delta_4$
		$w_{2,5} \longleftarrow w_{2,5} + \eta a_2 \Delta_5$
	1	$w_{1,4} \longleftarrow w_{1,4} + \eta a_1 \Delta_4$
		$w_{1,5} \longleftarrow w_{1,5} + \eta a_1 \Delta_5$

Como vemos, el algoritmo no es excesivamente complejo. **Desgraciadamente posee tres grandes fallos.** El primero es que **no tiene una condición de parada**, es decir, cuándo los pesos de las sinapsis son aceptables. El segundo es que este algoritmo **puede estancarse en los mínimos locales**, esta tara le viene de basarse en el método del descenso del gradiente. Por último, y comentado en la sección 2, **este algoritmo no tiene forma, ni de controlar, ni de medir el sobreajuste.**

3.2 Momentum.

No es propiamente un algoritmo, sino más bien una mejora que se introduce al algoritmo de retropropagación. Como vimos antes, uno de los problemas que tenía era que se podía estancar en un mínimo local. Con el fin de evitar esto, surge el *momentum*, que no consiste en más que introducir un sumando adicional a la hora de actualizar los pesos de las sinapsis, que tiene en cuenta la actualización realizada en la iteración anterior. Recordemos la fórmula de actualización de pesos anterior:

$$w_{ji} \leftarrow w_{ji} + \eta a_j \Delta_i$$

- Como vemos no se tiene en cuenta la actualización anterior, solo el peso.

Ahora, en una iteración n-ésima, la fórmula que tendremos sera esta:

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}^{(n)} \quad \text{donde} \quad \Delta wji^{(n)} = \Pi a_j \Delta_i + \alpha \Delta wji^{(n-1)}$$

De esta manera se evita que la retropropagación se estanque en mínimos locales.

3.3 Validación Cruzada.

Otro de los problemas que comentábamos antes del algoritmo de retropropagación, era el hecho de que no tenía una forma de controlar el sobreajuste. Para ello, combinando este algoritmo con la mejora de la sección siguiente (3.4 Parada temprana) se aporta una herramienta para evitar o intentar controlar el sobreajuste.

¿En qué consiste la validación cruzada? Antes de responder a esa pregunta, es necesario definir una serie de conceptos:

- *Epoch*: es una iteración de la fase de entrenamiento. Equivaldría a recorrer una vez todo el conjunto de entrenamiento, realizando los cálculos pertinentes.
- *Strip*: se refiere a un marcador que se activa tras un conjunto de *epochs*, por ejemplo en el artículo de usa un *strip* de 5, lo que significa que cada 5 *epochs* se activara este marcador.

Ademas se utilizan las siguientes medidas de error:

- *Squared error*:

$$E(o,t) = \sum_i (o_i - t_i)^2$$

o_i: valor obtenido en la sinapsis i-ésima de salida.

t_i: valor esperado en la salida.

- *Mean error* (media de los errores anteriores)

$$E = 100 \cdot \frac{o_{max} - o_{min}}{N \cdot P} \sum_{p=1}^{P} \sum_{i=1}^{N} (o_{pi} - t_{pi})^2$$

O_{max} → es el valor máximo que puede aparecer en la salida de la red.

O_{min} → es el valor minimo que puede aparecer en la salida de la red.

Una vez definidos los conceptos anteriores, y partiendo de la estructura de ficheros descrita en el apartado 2.2 planteada en [1], se utiliza el siguiente proceso para entrenar la red.

Se coge el conjunto de entrenamiento, y se va entrenando la red, y almacenando los *squared erro*r asociados a cada par entrada, salida. Cuando se acaba el conjunto, se alcanza un *epoch*, se realiza la media denominada **Etr**. Al alcanzar un *strip,* se recorre todo el conjunto de validación y se lanza sobre la red, calculando el error cuadrático medio producido. Dicho valor se denominara ***Eva.***

A continuación, siempre en el mismo *strip,* se realizan los siguientes cálculos:

1. Calculamos ***Eopt*** que vale el ***Eva*** mínimo registrado hasta ahora.
2. Calculamos GL para el *strip* actual, para ello dividimos el ***Eva*** del *strip* actual entre el ***Eopt***, le restamos 1 y multiplicamos por 100.

$$GL(t) = 100 \cdot \left(\frac{E_{va}(t)}{E_{opt}(t)} - 1 \right)$$

- Si el valor de **GL** para el *strip* actual es mayor que el strip (5 en caso del trabajo) se detiene el proceso de aprendizaje.

3. Por último calculamos un parámetro que representa el progreso de aprendizaje, ***Pk***. Para ellos dado un *strip* se cogen todas las medias de los errores calculados para el conjunto de entrenamiento, los **Etr**, y se dividen estos entre el mínimo de todos ellos por el parámetro *k*. Se le resta uno y se multiplica por 1000.

$$P_k(t) = 1000 \cdot \left(\frac{\sum_{t' \in t-k+1 \ldots t} E_{tr}(t')}{k \cdot \min_{\ldots} E_{tr}(t')} - 1 \right.$$

Con los parámetros anteriores y el criterio de parada temprana, se consiguen dos cosas: evitar el sobreajuste y dotar al algoritmo de retropropagación de una serie de parámetros de parada.

3.4 Parada temprana.

Este no es un algoritmo en sí, sino un conjunto de criterios de parada. La utilidad que tienen es que combinando estos criterios con el algoritmo de validación cruzada y usando ambos en el de retropropagación se consigue poner remedio a dos defectos de los que se hablo en la sección 3.1: el sobreajuste y la falta de criterios de parada.

La parada temprana dictamina que el algoritmo debe dejar de entrenar la red, cuando se cumpla una de las siguientes condiciones:

1. Se alcanza un número de *epochs* máximo.
2. Cuando GL(t) alcanza un valor superior al *strip*.
3. Cuando el parámetro Pk(t) en un *strip* es menor que 0.1.

Una vez alcanzada una de esas condiciones la red neuronal que se devuelve no es la actual, ya que ésta, por un motivo o por otro, no es la óptima.

La red que se devuelve es aquella de las anteriores, que poseía un error de validación mínimo, el *epoch* en el que se produce esto se denomina *relevant epoch.*

Debido a esto, y comentado en las conclusiones, es de suma importancia almacenar los distintos estados por los que va pasando la red durante la fase de entrenamiento. Esto fue un gran problema para mí durante la elaboración del trabajo, porque no existe nada ya programado, como una librería, que de soporte para ello.

4 Experimentación

En este apartado primero explicaré los problemas escogidos para la experimentación de forma teórica, y después expondré los resultados obtenidos. Pero antes, creo que es necesario aclarar algunos puntos.

En la sección de experimentación, escogí 5 ficheros del conjunto PROBEN: cancer1.dt, cancer2.dt, glass1.dt, card1.dt, horse1.dt. He intentado escoger ficheros que representan problemas lo menos "cercanos" entre ellos, para reflejar la gran adaptabilidad de las redes a cualquier tipo de problema

Se puede ver que he escogido dos ficheros que son de un mismo problema, cancer1 y 2. El motivo es que, como se explica en PROBEN, cada fichero tiene ejemplos distintos y entrenar una red con uno o con otro (según el propio articulo), podría llevar a una mejor o peor eficiencia. Con el fin de comprobar esto he tomado esta elección.

Por otro lado, antes de comenzar a explicar los problemas, me gustaría hablar de los parámetros estadísticos mostrados y medidos. Para cada fichero he realizado 10 ejecuciones sobre 10 redes distintas, así que para cada fichero almaceno los siguientes datos relevantes:

- Nombre del archivo perteneciente al conjunto PROBEN.
- Nombre de la red y su topología.
- Constante de aprendizaje usada.
- Constante usada para el momentum.
- O_max y O_min son los valores máximos y mínimos que podemos encontrarnos en el problema. En la implementación de mi trabajo he añadido una mejora que permite no especificar dichos parámetros (dejándolos a 0 los dos) y la propia red conforme se entrena los calcula. Esto podría acarrear un estancamiento en un mínimo o máximo local aunque en problemas en los que no se conocen dichos parámetros puede volverse muy útil.
- *Outputs were initialized* indica si los dos parámetros anteriores se especificaron o los calculó la red.
- El número de ejemplos en el conjunto de entrenamiento.
- El error cuadrático medio mínimo obtenido en la fase de entrenamiento.
- El número de ejemplos en el conjunto de validación.
- El error cuadrático medio mínimo obtenido en la fase de validación.
- *Overfit* es el grado de sobreajuste.
- Número de épocas totales.
- Número de épocas relevantes.
- El número de ejemplos en el conjunto de test.
- El error cuadrático medio obtenido en la fase de testing.
- Número de ejemplos mal clasificados en el *testing*.

Una vez que obtengo todos esos valores para cada uno de los 10 archivos, creo un resumen mayor, donde expongo los valores medios más relevantes

- *Training Set Mean*: error medio del conjunto de entrenamiento.
- *Training Set Stdev*: desviación típica del conjunto de entrenamiento.
- *Validation Set Mean*: error medio del conjunto de validación.
- *Validation Set Stdev*: desviación típica del conjunto de validación.
- *Test Set Mean*: error medio del conjunto de testeo.
- *Test Set Stdev*: desviación típica del conjunto de testeo.
- *Test Set Clasification Mean*: el error medio de ejemplos mal clasificados en porcentaje.
- *Test Set Clasification Stdev*: la desviación tipica de los ejemplos mal clasificados.
- *Overfit Mean* y *Stdev*: media y desviación del sobreajuste.
- *Total Epochs Mean* y *Stdev*: media y desviación de las *epochs* totales.
- *Relevant Epochs* Mean y *Stdev:* media y desviación de las *epochs* relevantes.

La elección de estos datos fue debida a que es la misma usada en PROBEN.

En la siguiente sección, hablaré sobre lo que cada fichero seleccionado, para experimentar representa y la estructura de la red obligatoria que se requiere para usarlos.

En la última expondré, por un lado para cada fichero los datos antes indicados y, al final, el resumen de los mismos.

4.1 Descripción de los experimentos

- Cancer1 y cancer2 – Estos dos ficheros representan el diagnostico de cáncer de pecho. Dado un tumor, trata de clasificarlo como benigno o maligno basándose en varios parámetros de las células, observables con un microscopio: espesor del grupo de celular, la uniformidad del tamaño de las células y su forma, la cantidad de adherencia marginal y la frecuencia con la que se observan núcleos descubiertos, etc. La red requerida para estos dos experimentos debe tener obligatoriamente 9 neuronas de entrada y dos de salida. Además los dos conjuntos poseen 699 ejemplos, entre entrenamiento, validación y testeo.

- Glass1 – Representa un problema de clasificación de cristales, la motivación de este *benchmark* es por motivos forenses y criminólogos. Como parámetros de entrada se le pasa a la red el índice de refracción y los resultados de un análisis químico de astillas de cristal. El número de neuronas en la entrada es obligatoriamente de 9 y en la salida de 6. El conjunto de ejemplos posee 214 casos, entre los que se encuentran los de entrenamiento, validación y testeo.

- Card1 – Presentan situaciones en las que se aprueba o no la decisión de conceder un crédito a un cliente. Las entradas de la red no se mencionan en PROBEN, por motivos de confidencialidad, pero el número de neuronas en la entrada tiene que ser 51 y en la salida 2. Además, este conjunto posee 690 ejemplos, entre entrenamiento, validación y testeo.

- Horse1 – En este conjunto se plantea el destino de un caballo que posea un cólico. Los resultados de un examen realizados por un veterinario son las entradas que permiten predecir si el caballo sobrevivirá, morirá o se le aplicará la eutanasia. El número de neuronas en la entrada serán 58 y en la salida 3. El número de ejemplos totales disponibles es de 364.

4.2 Resultados obtenidos

En esta sección no pondré una imagen de los resultados obtenidos ya que es tan grande que no cabe en una página, en su lugar se encuentran disponibles en la carpeta anexo. Es recomendable tener el archivo de cada sección abierto ya que explicare partes muy detalladas de los mismos.

4.2.1 – Cancer1

- Para este fichero la primera red que hice tenía una topología simple de una capa de entrada y solo otra de salida. En principio, obtuve pocos fallos de clasificación, un entrenamiento en muy pocas épocas y un sobreajuste relativamente alto. Las siguientes pruebas que realicé fue debida a que no sabía cómo podían o cuánto podían afectar las constantes: tanto la de aprendizaje, como la del momentum. Por lo tanto, las siguientes cuatro redes tenían todas la misma topología (hasta la red "red_cancer_5.txt"). Por un lado, se observa que conforme la constante momentum va disminuyendo, el número de épocas necesarias para el entrenamiento, y la relevante, aumentan. Por otro lado, el sobreajuste disminuye, y teniendo en cuenta que en comparación al incremento de épocas disminuye más, todo indica que un momentum pequeño es más eficiente. La prueba definitiva se da cuando se observan el número de ejemplos mal clasificados, ya que se reducen drásticamente, es decir, prácticamente a la mitad.
- Por otro lado, la constante de aprendizaje tiene una curva de eficiencia distinta, mientras que la del momentum podría describirse como una línea ascendiente (en eficiencia), conforme la constante se hace más pequeña, la de aprendizaje describiría una campana. Si observamos las redes 5, 4 y 2, vemos que cuando el valor que toma el parámetro es medio, la eficiencia es mayor, el entrenamiento requiere de menos épocas y hay menos fallos en la clasificación. Aunque cabe mencionar que el sobreajuste es mayor. Debido a esto, para el resto de pruebas escogí un valor para esta constante pequeño, ya

que es el que posee un sobreajuste medianamente malo, no es el mejor, pero tampoco el peor con la ventaja de que clasifica mal pocos ejemplos.

- Las siguientes dos pruebas (6 y 7), fueron realizadas para ver cómo se comportaba la red cuando tenía muchas neuronas y muchas capas intermedias (tanto capas regulares como no regulares). El resultado fue relativamente desastroso, aunque el sobreajuste es mínimo, el número de ejemplos mal clasificados es prácticamente la mitad. Además, que alcanzaron el máximo de épocas posibles y el entrenamiento tardo mucho.
- Los últimos tres ejemplos (8, 9 y 10), estaban enfocados a reducir el tiempo de entrenamiento de las redes y disminuir el número de fallos a la hora de clasificar. Las topologías son parecidas, con una o dos capas intermedias, una vez con muchas neuronas y otra con pocas. El resultado obtenido fue el esperado, el número de ejemplos mal clasificados se reduce casi a la mitad que el obtenido por las redes 6 y 7, pero a un gran coste, el sobreajuste. Obtienen un sobreajuste muy grande, casi tanto como el que se obtuvo con la estructura de los primeros 5 experimentos, sólo que en esos el número de ejemplos mal clasificados era mínimo.
- **Conclusión:** Para este problema las redes pequeñas, con un valor adecuado en las constantes obtienen un valor de sobreajuste relativamente bajo, se entrenan rápido y su tiempo de computación es mínimo y el número de ejemplos mal clasificados es, en el peor de los casos, del 19 % y en el mejor del 9%. Tomando un valor medio creo que el sobreajuste sería aceptable para tratar ejemplos reales. Por otro lado, las grandes redes fallan como en el 50% de los ejemplos, aunque poseen un sobreajuste próximo a 0. Aunque no sirven de mucho si clasifican tan mal. Las redes "intermedias" tienen buenas prestaciones, pero son ligeramente peores que las pequeñas. **Debido a todo esto, creo que las mejores redes para este problema son las más básicas, siempre que se haga un ajuste de los parámetros.**

4.2.2 – Cancer2

- Parte del objetivo de usar este fichero era comparar si realmente variaban tanto los resultados con los del fichero "cancer1.dt".
- En las dos primeras redes uso una topología básica igual que en el fichero anterior, en este caso sólo varió la constante de aprendizaje, pero obtengo unos resultados muy parecidos. Cabe destacar que parece que en este fichero la sensibilidad del sobreajuste a la variación de esta constante es menor, pero no olvidemos que parte de la red tiene un componente aleatorio, así que ésta diferencia podría no ser real, aunque como valor medio también se obtiene un sobreajuste medio menor que el obtenido en cancer1.
- En la siguiente red, la 3, intenté probar a ver cómo se comportaba una red de muchas capas ocultas con este fichero. Los resultados obtenidos son curiosos, aunque no son del todo fiables. Para empezar no posee sobreajuste. Esto es posible, pero es una coincidencia. Si observamos las épocas transcurridas vemos que la red estaba llegando casi al máximo y justo antes paro porque el

parámetro Pk cumplió la condición establecida; debido a esto es probable que esta red se haya estancado en un mínimo local. Aunque esto es sólo una teoría que habría que comprobar. Aún así, cabe mencionar que el número de fallos es muy alto.

- La siguiente red, la 4, pretendía comprobar la eficiencia con una red de una capa intermedia, recordemos que antes con las de este tipo obteníamos resultados ligeramente peores que los obtenidos con las redes mas básicas. En este caso ocurrió lo mismo el rendimiento medio comparado con el rendimiento medio de las mas pequeñas fue peor, sobretodo por el sobreajuste. Debido a que realmente fue por eso, en el siguiente ejemplo pensé en introducir más capas intermedias, para ver si se podía reducir. En la red 5, vemos que es así, pero tarda muchas más épocas en entrenarse y el número de fallos es ligeramente mayor. Aún así, aparece una mejora. Como parecía que las redes con capas intermedias podían, en este caso, tener mejor eficiencia que las más pequeñas, probé creando la red de los ejemplos 6, 7, 8 y 9 las cuales tienen todas la misma topología y trato de reducir el overfitting y el número de fallos de clasificación, modificando el parámetro del momentum. Los resultados obtenidos son caóticos, ya que no se ve una clara mejora en la clasificación de los errores, incluso podría decirse que empeora un poco. Donde cabe destacar una mejora es en el sobreajuste que, con un valor medio para el momentum, es tan bajo como el de las redes con sólo una capa de entrada y otra de salida. De hecho, prácticamente, posee la misma eficiencia que la red 1.
- La última red de este apartado fue un intento de probar si añadiendo muchas capas ocultas, igual que antes, pero cambiando las constantes, se podía obtener unos valores mejores. Pero los resultados son los mismos que los obtenidos en la red 3.
- **<u>Conclusiones:</u> Con respecto a las redes creadas para este problema vemos que, igual que antes, las más pequeñas son las realmente más eficientes si se afinan los parámetros, pero también las que poseen una capa oculta con muchas neuronas pueden llegar a ser competitivas con los parámetros adecuados. Aún así, en los dos casos el sobreajuste es relativamente alto.**
  Con respecto al fichero "cancer1.dt", el sobreajuste medio es menor y el número de fallos medio también. Esta diferencia realmente creo se podría adjudicar a los archivos y no realmente a las redes. Sería interesante crear una red con uno de los ficheros y, luego, hacer el testing con el otro, para ver los resultados obtenidos.

4.2.3 – Glass1

- Lo que más llama la atención de estas pruebas es que todas las redes obtienen un número de ejemplos mal clasificados iguales. Esto se debe principalmente a que casi todas se quedan con la red de la misma época relevante. Y

aunque pudiera parecer un error de la red no lo es, ya que auque todas fallen el mismo número de veces el sobreajuste, dato muy importante, si que varía mucho.

- En las redes 1, 2, 3 y 7 la topología es de lo más básica, una capa de entrada y una de salida. En el ejemplo 1, uso una constante de aprendizaje mucho mayor que en el 2 (el doble para ser exactos), y obtengo un sobreajuste relativamente alto. Para bajarlo en el dos, lo reduzco a la mitad, pero sorprendentemente para mi el valor del overfitting aumenta. Debido a esto, en el ejemplo 3, mantengo la constante de aprendizaje y aumento el momentum, obteniendo casi el mismo overfitting. Por lo tanto ¿el sobreajuste es debido al valor que toma la constante de aprendizaje? Para responder a esta pregunta realizo luego la prueba numero 7, donde uso la misma red que en la prueba 2, con el fin de ver si efectivamente vuelvo a obtener el mismo sobreajuste (y entonces podemos afirmar que se debe a la constante de aprendizaje) o se obtiene un valor distinto. El resultado fue prácticamente el mismo sobreajute.
- En los experimentos 4, 5 y 6, pruebo a ver cómo se comportan para este fichero las redes con muchas capas intermedias y bastantes neuronas en cada una. Los resultados obtenidos son bastante buenos, la topología no varía de red en red, en su lugar varió el momentum, conforme lo incremento (hasta llegar a uno altísimo) veo que el sobreajuste va disminuyendo de mucho. Al fin,al para su máximo valor, obtengo un sobreajuste casi de cero y un número de fallos de clasificación igual que el resto de redes, con lo cual hasta ahora esta parece la mejor opción. Lo único malo que tiene que computar el máximo de épocas posibles, con lo que el proceso de entrenamiento es un poco lento.
- En la red 8, ése una topología un tanto peculiar, ya que me pregunté qué pasaría si la red tuviera la capa de entrada, una neurona en la capa oculta y la salida. Los resultados son malísimos, el valor de sobreajuste es muy alto.
- En la red 9, la topología fue la misma que en la red 8, sólo que con 20 neuronas en la capa intermedia. Vimos que se redujo un poco el sobreajuste, con lo cual quedó claro que esta topología no funcionaba bien.
- Debido a que hasta ahora la mejor topología había sido la que tenía muchas capas ocultas con muchas neuronas, en la red 10, construí una red monstruosa. Aunque los resultados fueron muy buenos, quedo claro que era mejor usar una red grande con capas ocultas no regulares y que el número de neuronas fuera descendiendo, como en la red 6.
- **<u>Conclusiones:</u> Para el problema planteado en este fichero la red ideal es aquella que posee varias capas ocultas, no regulares, que vayan decrementando el número de neuronas, aunque nunca por debajo del número de neuronas de la capa de salida.** De esa manera, obtenemos una red con un número de fallos bajos y un sobreajuste prácticamente nulo. Con respecto a las redes más pequeñas, no son eficientes, ya que poseen un valor de overfitting demasiado grande. Con respecto a los parámetros el mejor valor que puede tomar la constante de aprendizaje es 0.8, mientras que la del momentum conforme más grande, a priori, parece que mejor.

4.2.4 – Card1

- Como siempre empecé realizando los dos primeros experimentos con dos redes de una capa de entrada y otra de salida. Para la red 1 y la red 2 sólo varié el momentum, las dos tienen el mismo número de fallos y de épocas, solo que en la red 2, al aumentar el momentum, también lo hace el sobreajuste. En general las dos redes tienen un sobreajuste relativamente alto.
- En la red 3, añadí una capa oculta con muchas neuronas, el resultado fue desastroso ya que, aunque el número de fallos fue sólo 1, el sobreajuste se disparó a un nivel inaceptable.
- Como por lo general, al añadir muchas capas ocultas con muchas neuronas, el valor del sobreajuste suele decrementarse mucho y en el apartado anterior se obtuvieron muy pocos fallos y un sobreajuste muy alto en la red 4, planteé una red con esa idea. Los resultados fueron los esperados con respecto al overfitting, pero con respecto a los fallos, se incrementaron hasta clasificar mal un 30% aproximadamente de los ejemplos.
- Debido a los resultados obtenidos antes, cambié de estrategia, en la red 5, volví a plantear una red con sólo una capa oculta, pero esta vez muchas más neuronas que las de la red 3. El sobreajuste fue otra vez altísimo.
- Así que en la siguiente red, la 6, reduje el número de neuronas de la capa oculta, para ver si el overfitting también descendía, y así fue. Debido a esto decidí escoger un valor medio de neuronas para la capa oculta.
- En las redes 7, 8 y 9, usé el mismo número de neuronas en la capa intermedia y varié los parámetros para intentar reducir el sobreajuste. El único resultado que cabe mencionar fue el de la red 8, que al tener el mayor valor en la constante de aprendizaje de los tres obtuvo el menor sobreajuste y muy pocos fallos. Aun asi el sobreajuste seguia siendo muy grande. Habria sido interesante probar con valores mas grandes, porque creo que se habría alcanzado un buen rendimiento.
- En el último experimento, el 10, en lugar de seguir por el razonamiento de las redes 7, 8 y 9 quise probar otra cosa. En la red número 4 había obtenido un sobreajuste de 0, sólo que el número de fallos había sido muy grande. Con el fin de decrementar ese parámetro, intentando dejar igual el overfitting, creé la red 10, la cual tenía menos capas que la red 4, pero muchas neuronas intermedias. El resultado de mi teoría no coincidió con mi predicción, ya que, no sólo tuve los mismos fallos de clasificación, sino que además esta vez el sobreajuste fue muy grande.
- **<u>Conclusiones:</u> Para este problema la red más adecuada para utilizar es aquella que tiene una estructura simple, una capa de entrada y otra de salida. El valor de sobreajuste es aceptable y el número de fallos pequeño.** Quizás si se usase una red con una capa oculta, con un número de neuronas medio, como 25, y se afinarán las constantes de aprendizaje y de momentum, y se podría alcanza el mismo rendimiento, pero es sólo una hipótesis, ya que no he podido realizar más pruebas para este fichero.

4.2.5 – Horse1

- Para este archivo empecé creando una red de sólo una capa de entrada y otra de salida. El resultado obtenido fue un sobreajuste alto y un 30% de ejemplos mal clasificados.
- Debido a los resultados anteriores, elegí introducir una capa oculta con bastantes neuronas, la red 2 obtuvo un sobreajuste aún mayor y un mayor número de fallos.
- Aunque el añadir una capa no dio mejores resultados en la red 3, introduje muchas más capas, esta vez con muchas más neuronas en cada una. El sobreajuste obtenido fue cercano a 0, pero el número de fallos fue como del 66%. Debido a esto, volví a intentar reducir el número de fallos, creando redes con muchas neuronas en la capa oculta.
- Para corroborar esa teoría, creé las redes 4 y 5. Las dos tienen una sola capa oculta, pero la primera tiene muchísimas neuronas, mientras que la segunda tiene muy pocas. La red 4 obtuvo un sobreajuste razonable y falló como el 50% de los ejemplos. Por otro lado, la red 5 falló un poco menos, pero su sobreajuste fue mucho mayor.
- Para comprobar si efectivamente tener muchas neuronas en la capa oculta podía llevar a una buena eficiencia, realicé el experimento número 10, la red 10 en este caso tenía más neuronas que la red 4. Obtuvo un menor sobreajuste, pero también un mayor número de fallos.
- En este punto decidí volver a intentar buscar la red óptima entre las que sólo tenían una capa de entrada y otra de salida, ya que el número de fallos era bastante menor y el overfitting, aunque mayor, lo era de poco. Con este fin, cree las redes 6, 7, 8 y 9. Todas con la misma topología. En las dos primeras comprobé cómo afectaba la constante de aprendizaje, al incrementarla obtuve menos fallos y el menor sobreajuste hasta ahora. Después, en la red 8, manteniendo el mismo valor para la constante de aprendizaje, incrementé la del momentum, el resultado fue un menor sobreajuste, pero muchísimos más fallos. Tras esto, descarté la idea de incrementar el momentum. Habría estado bien probar valores más pequeños para este parámetro, pero en su lugar, en la red 9, lo que hice fue incrementar la constante de aprendizaje, obtuve un overfitting ligeramente mayor, pero aceptable y definitivamente muy pocos fallos, aproximadamente un 30%.
- **<u>Conclusiones:</u> Para este problema las redes más simples, con una capa de entrada y otra e salida, son las más eficientes. Creo que incrementando lo suficiente la constante de aprendizaje y decrementando la del momentum se alcanzaría una eficiencia óptima.** Aunque no he podido comprobarlo, en los 10 experimentos de este trabajo. Quizás también las redes con una capa oculta y muchísimas neuronas intermedias podrían dar buenos resultados, pero el tiempo de entrenamiento de estas es bastante mayor que el de las primeras, por lo que me quedaría con estas.

5 Conclusiones y trabajo futuro

- La principal conclusión que he podido sacar de todo este trabajo ha sido que las redes neuronales están en una fase aún muy poco desarrollada, especialmente en lo que a la persistencia de las mismas se refiere (para mí esto fue un gran problema a la hora de realizar el trabajo). No existe tampoco ninguna estandarización para la representación de las redes, esto es algo que llama la atención. Pero por otro lado me ha sorprendido la flexibilidad de las redes, la cantidad de problemas que pueden abarcar, lo relativamente simples que son de crear y entrenar, aunque sin duada todo el trabajo expuesto en [1] ayuda muchísimo a este propósito.

Además es remarcable la potencia que poseen, como con un entrenamiento realmente "corto", luego son capaces de inferir datos con un porcentaje de fiabilidad medible. Aunque esto es algo que sostengo un poco de forma teórica, ya que no he tenido la oportunidad de usar una red entrenada en el terreno "real".

- Por otro lado, y retomando un poco lo que antes decía de la estandarización, si las redes son tan potentes por que no existe una estandarización para su representación y almacenamiento. Esto creo que resolvería muchos problemas de compatibilidad de los que se hablan en el documento [1] PROBEN.

- Sobre las conclusiones experimentales, ha quedado claro que cada problema posee unos valores para las constantes y una topología propia. Por lo tanto es categórico cuando se aborda un problema realizar un estudio con varias redes para establecer los mejores valores para cada cosa. En este trabajo se han usado 10 ejecuciones por fichero, en escenario más centrado creo que 10 serían muy pocas. Durante el desarrollo del trabajo hubo veces que me habría gustado hacer más que ese número para comprobar teorías sobre que red sería mejor que otra y porqué.

- En conclusión, creo que este trabajo me ha dejado claro que cada problema tiene sus propias características y requiere de una red distinta. Por otro las redes neuronales y todo lo que gira a su alrededor esta aun en los albores de su desarrollo y personalmente creo que son una herramienta simple pero muy eficiente. Al terminar este trabajo, me ha parecido interesante para futuras ampliaciones, desarrollar el punto del almacenaje de redes, ver algún otro algoritmo de entrenamiento, para ver si su eficiencia es mejor que el de retropropagación, pero sobretodo el paralelismo a nivel hardware del que hablo en las ventajas de las redes neuronales.

Referencias bibliográficas

1. Lutz Prechelt, PROBEN1 – A set of Neural Network Benchmark Problems and Benchmarking Rules, Technical Report 21/94, 1994.
2. S. Russel, P. Norvig. Inteligencia Artificial: un enfoque moderno, Capítulo 20, Prentince Hall, 3ª ed. (2010).
3. Guía de autores para la serie Lecture Notes in Computer Science, de Springer http://www.springer.com/librarians/e-content?SGWID=0-113-12-558799-0
4. D. Balbontín Noval, F. J. Matín Mateos, J. L. Ruiz Reina, Introducción a las Redes Neuronales.