Práctica de curso de IA2
Redes Neuronales

Domingo Llorente Rivera
Salvador Jesús Romero Castellano

Introducción

En esta memoria describimos el desarrollo, diseño y resultados de nuestra práctica de curso.

En la primera sección describimos cómo hemos tomado las fotografías y cómo las hemos tratado, además de cómo colocarlas en una estructura de directorio para que el sistema averigüe automáticamente cuál es la salida que corresponde a cada fotografía.

En la segunda sección, se describe el funcionamiento de la interfaz de usuario, y cómo experimentar con esta.

En la tercera sección, se describe el funcionamiento de la red neuronal en detalle.

Por último, en la cuarta sección, se muestran los resultados que hemos obtenido experimentando con el sistema.

1. Toma y organización de fotografías

Realización de las fotografías

Todas las fotografías de este trabajo han sido realizadas por nosotros. Además de las fotos solicitadas sobre lenguaje de signos hemos añadido fotografías de señales de banderas. Cada letra del alfabeto se ha fotografiado varias veces con variaciones en la posición y ángulo, para intentar conseguir que las fotos abarquen un conjunto los más amplio posible. Las fotografías del lenguaje de signo han sido realizadas con guante negro, y el fondo ha sido borrado con un programa de edición para que sea blanco contrastado. Las fotografías de señales de banderas no han sido retocadas, para comprobar hasta qué punto el preprocesado es importante.

Formato de las fotografías

Las fotografías han sido tomadas en blanco y negro, en resolución VGA y con formato JPG. Para la optimización del programa se han convertido las fotografías a formato PGM, y una resolución de 43x32 para no perder la relación de aspecto de las originales

Hay varios tipos de compresión .pgm, en este trabajo se ha usado ASII.

```
P2
# Created by IrfanView
43 32
255
255 255 255 255 255 255 255 255 255 253 255 255 254 254 255 253
255 254 255 255 253 254 255 254 255 252 255 253 255 253 255 254
255 253 255 255 255 250 255 255 255 255 255 255 255 255 255 255
254 254 255 255 255 253 253 255 254 250 255 255 255 255 253 255
249 255 251 248 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 254 249 250 255 255 255 247 246 255 254
```

Para que el programa funcione bien, la cabecera de los archivos ha de disponer de una forma similar a la captura anterior:

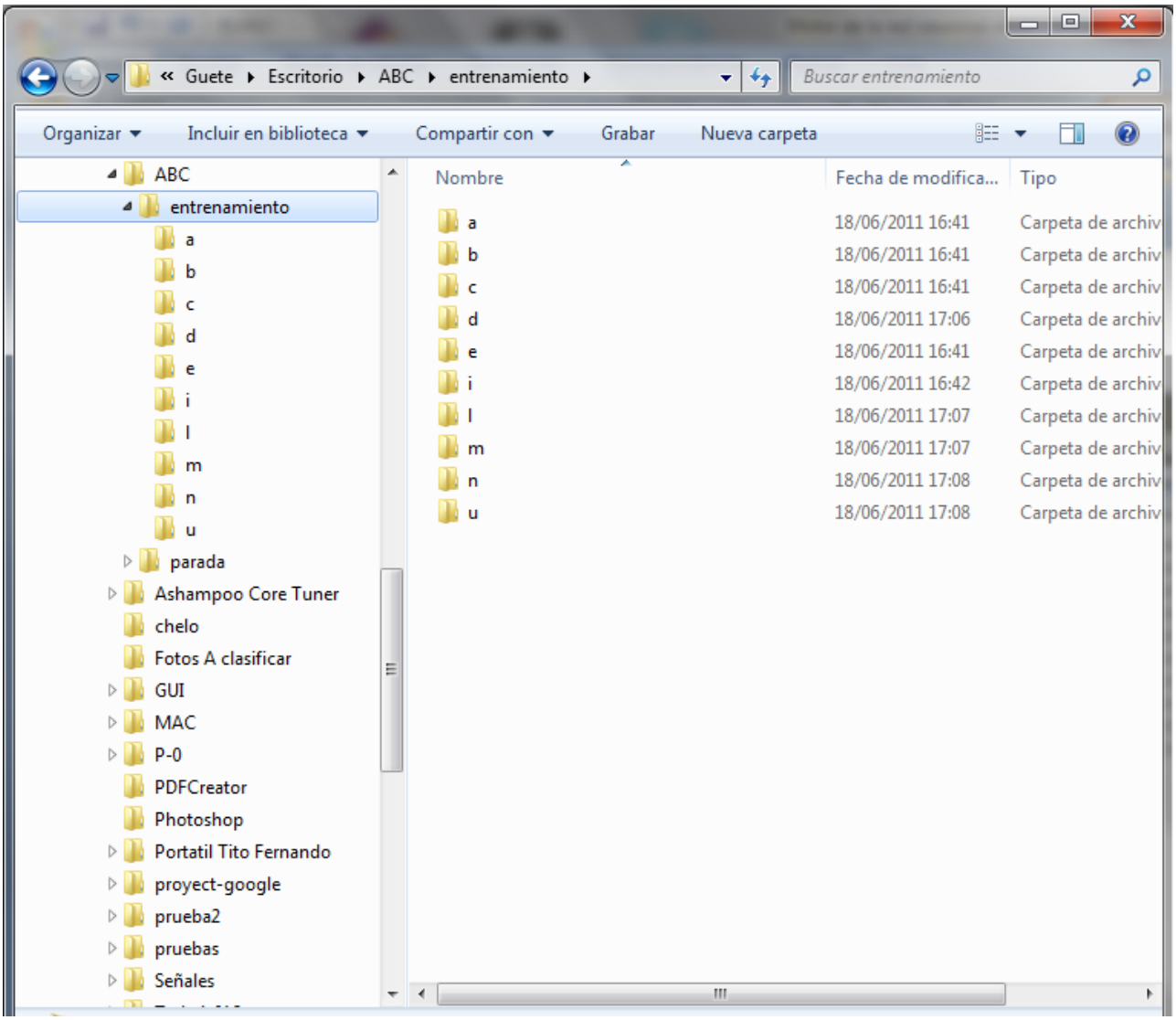
- Dos líneas de cabecera.
- Dimensión de la fotografía.
- Datos.

Esto es debido a que el ***número de entradas a la red neuronal se calcula de forma automática*** en la lectura de los ficheros, para así poder trabajar con distintos ficheros sin tener problemas de configuración.

Organización de las fotografías

Este punto es muy importante, ya que la organización de las fotografías influye en la salida de la red.

Las fotografías han de estar situadas dentro de una carpeta, raiz. Esta carpeta a su vez contendrá un conjunto de carpetas clasificando el contenido de estas. Como se puede ver en la siguiente captura:



En esta fotografía, se muestra la estructura de las carpetas, para el abecedario. En la que se puede apreciar como la carpeta entrenamiento contiene las distintas carpetas a, b, c, d… Cada una de las carpetas contendrá fotografías en formato PGM.

El motivo de organizar la estructura de carpetas de esta forma es para facilitar al usuario que maneje el programa, como ***por ejemplo para cambiar de conjunto de entrenamiento***. De esta forma, el programa lee el número de subcarpetas y configura el número de salidas de la red. Además, es recomendable que las carpetas tengan el nombre de lo que representan las fotos que hay en su interior, ya que el programa toma el nombre de la carpeta para informar sobre cuál es la clasificación que se le da a una foto.

2. Uso del sistema

Casos de uso del sistema

El sistema permite una amplia variedad de funciones para el usuario:

- *Configuración de los parámetros de una red*: Permite configurar una red según los parámetros de factor de aprendizaje, factor de momento, peso mínimo, peso máximo y numero de capas con neuronas ocultas.
- *Cargar un conjunto de entrenamiento:* Permite carga una carpeta donde se encuentre un conjunto de entrenamiento estructurado adecuadamente por carpetas, como se ha visto anteriormente. La realización de este caso de uso permite realizar el caso de uso "Entrenar una red".
- *Cargar conjunto de entrenamiento como condición de parada*: Permite carga una carpeta, preferiblemente con fotos distintas a la cargada en el caso de uso anterior, donde se encuentre un conjunto de entrenamiento estructurado adecuadamente por carpetas, como se ha visto anteriormente. La realización de este caso de uso permite realizar el caso de uso "Entrenar una red, según criterio de parada por tanto por ciento de aciertos".
- *Entrenar una red*: Permite entrenar una red, según los parámetros configurados. Un número de veces finitos indicados por el campo "Iteraciones de entrenamiento", se verá más adelante.
- *Entrenar una red, según criterio de parada por tanto por ciento de aciertos*: De igual formar que el caso de uso anterior permite entrenar una RedNeuronal, pero con la diferencia de que esta vez el criterio para la finalización del entrenamiento será: fin del numero de iteraciones o tanto por ciento de aciertos acertados.
- *Clasificación de una fotografía en la red*: Este caso de permite seleccionar una fotografía y mostrar la predicción de clasificación en la red.
- *Guardar el estado de una red tras su entrenamiento*: Permite guardar el estado de una red que ya haya sido entrenada. En formato ASII, se recomienda guardar con extensión .txt. El resultado de este caso de uso es un fichero con información relevante para poder cargar el estado de una base de datos.

```
1377 5
1.0012286 0.47712687 0.3181192 -0.3696541 0.4665513 -0.35107476 -0.093841955 0.44041198 -0.3778265 0.09984849
1.0849193 0.74818337 0.6617811 -0.45264012 -0.077437624 -0.429436 -0.3153827 -0.21934961 0.04432904 -0.418848(

Conocimiento
B
0.9 0.0 0.0 0.0 0.0
I
0.0 0.9 0.0 0.0 0.0
J
0.0 0.0 0.9 0.0 0.0
K
0.0 0.0 0.0 0.9 0.0
R
0.0 0.0 0.0 0.0 0.9
```

Las tres secciones del archivo de configuración: Rojo: entradas y salidas; Negro: estado de la RedNeuronal; Verde: Valores de predicción de la red.

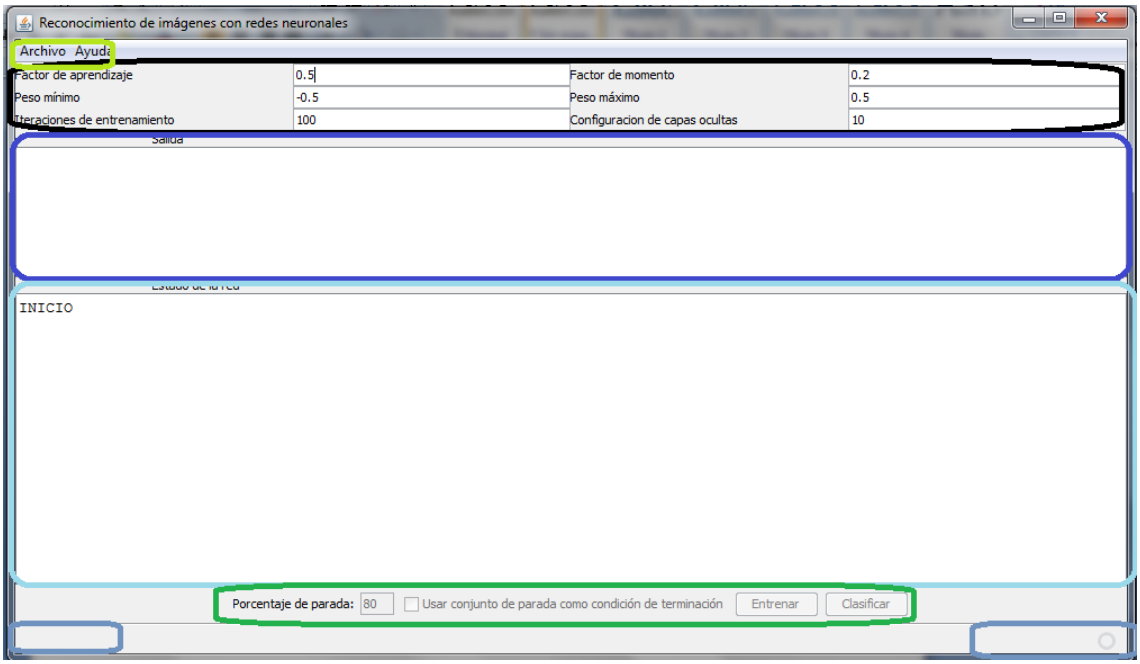
- *Cargar el estado de una red ya entrenada:* Carga el estado de una RedNeuronal entrenada. Para poder ejecutar este caso necesario haber ejecutado antes el caso de uso anterior. Se recomienda que antes de cargar un fichero la configuración de la red sea la misma.

Interfaces de Usuario

La interfaz de usuario presenta una interfaz conveniente para configurar y experimentar con el motor de red neuronal. Hemos desarrollado el motor de red neuronal como una librería, empaquetada un un archivo jar, y hemos diseñado la interfaz para que utilice dicha librería, procurando mostrar todas sus funcionalidades.

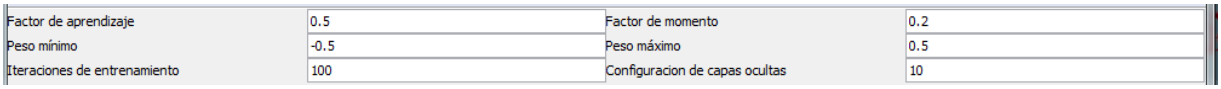
En este apartado trataremos como usar la interfaces de usuario, como configurar la red, y como interpretar los datos de salida.

Identificación de los distintos elementos de la interfaz



En negro: Parámetros para configurar la red; Azul: Información sobre las operaciones y salidas del sistema; Verde: Botones y controles de la aplicación.

Configuración de los parámetros de la red



En este aparado se verá como se realiza la configuración de una RedNeuronal.

Los elementos: Factor de aprendizaje, Factor de momento, Peso mínimo y Peso máximo han de ser elementos tipo float comprendidos entre 0-1. Estos numero determinan la configuración y cuyos significado ya se ha explicado anteriormente.

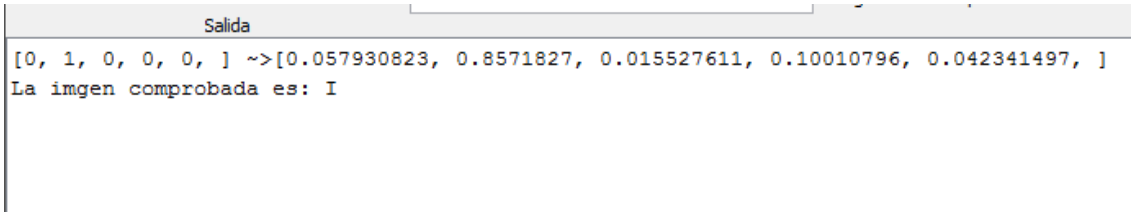
Configuración de capas ocultas. De tipo integer. Indica el número de capas ocultas y neuronas en cada capa. En esta caso el numero 10 indica el numero de neuronas ocultas. Si queremos tener más de una capa, será necesario colocar ';' entre los numero.

Es decir si queremos tener dos capas con 5 elementos en una y 10 en la otra. Deberíamos de poner “5;10”. De la misma forma si se quiere tener 5 capas con diversos elementos: 6;7;3;10;12. Para el caso particular de no tener ninguna capa oculta simplemente no poner nada. No hay limitación en cuanto al número de capas que se quiera poner.

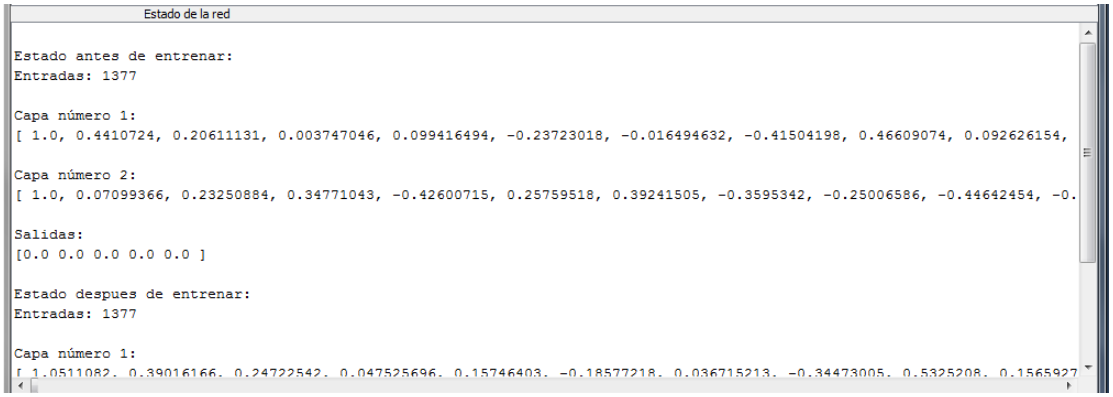
Iteraciones de entrenamiento: tipo Integer. Indica el número de iteraciones a realizar para entrenar el conjunto de neuronas.

Información

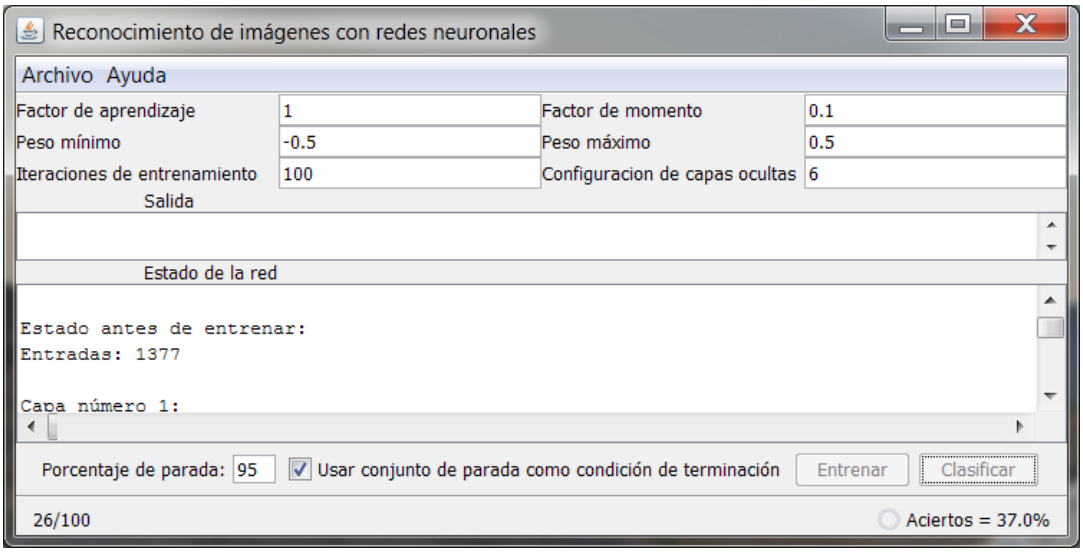
Salida: Muestra la información sobro una fotografía clasificada. Muestra el objetivo y la salida de las neuronas. Además de una breve información sobre qué tipo de letra sería si la puede clasificar.



Estado de la red. Muestra información sobre el estado de las neuronas antes y después del entrenamiento, o al cargar una configuración en la red.



Información del estado del proceso: Una vez pulsado el botón “Entrenar”, se verá en el siguiente paso. Se mostrará información relativa al estado del proceso en la parte inferior de la pantalla

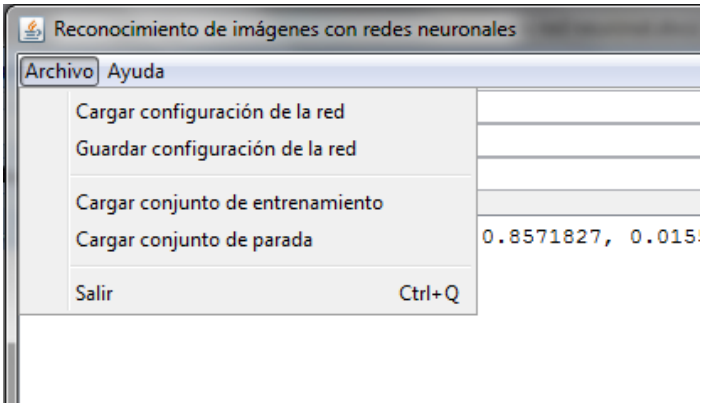


Captura de pantalla de un proceso de entrenamiento. Se puede observar en la parte izquierda de la barra de estado las iteraciones cumplidas y el porcentaje de aciertos sobre el conjunto de parada.

Botones

- **Entrenar**: Situado en la parte inferior de la ventana. Permite entrenar una red, un número de iteraciones con la configuración actual, en pantalla. No será posible realizar un entrenamiento si antes no se ha elegido un conjunto de entrenamiento. Con la estructura de directorios adecuada.
- **Clasificar**: Situado en la parte inferior de la ventana, al lado del botón "Entrenar". Nos permite clasificar una fotografía, después de haber entrenado una red o cargado una configuración anterior. Esta clasificación será mostrada por pantalla en la parte de salida mencionada anteriormente.
- **Tic Usar conjunto de parada**. Situado al lado del botón "Entrenar". Nos permite realizar el caso de uso "*Entrenar una red, según criterio de parada por tanto por ciento de aciertos"* al pulsar el botón "Entrenar". Antes de ello es necesario indicar el tanto por ciento de aciertos en "Porcentaje de parada". Nota esta opción estará desactivada hasta que configuremos un conjunto de parada en "Cargar conjunto de parada". Se verá más adelante.

Desplegable archivo:



Cargar Configuración de la red: Esta opción está disponible desde que se lanza el programa. Permite realizar el caso de uso "*Cargar un conjunto de entrenamiento".*

Guardar configuración de la red: Lanza el caso de uso "Guardar un conjunto de entrenamiento". Esta opción solo estará disponible una vez se haya lanzado el caso de uso anterior o se haya entrenado la red.

Cargar conjunto de entrenamiento: Lanza el caso de uso *"Cargar un conjunto de entrenamiento".* La realización de este caso de uso permite poder Entrenar

Carga conjunto de parada: Lanza el caso de uso "Cargar conjunto de entrenamiento con condición de parada". La realización de este caso de permitirá hacer el caso de uso "Entrenar una red, según criterio de parada por tanto por ciento de aciertos".

3. Motor de la red neuronal

En esta sección se explica de forma aproximada el funcionamiento de la red neuronal. El motor de red neuronal se ha implementado como una librería aparte, para que se pueda usar con otros sistemas si se desea. La librería está en un .jar que usa la interfaz, y tiene su documentación javadoc correspondiente.

Configuración de la red

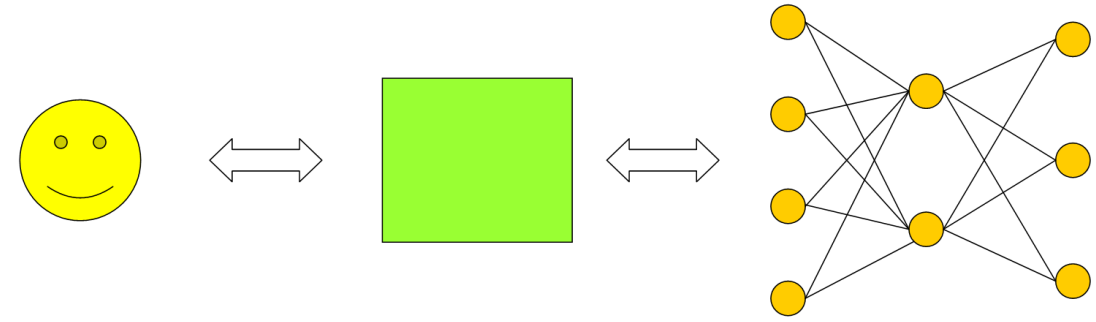
La red neuronal admite los siguientes parámetros de configuración: coeficiente de entrenamiento; coeficiente de momento; los pesos mínimos y máximos para la asignación aleatoria inicial de los pesos iniciales; el número de salidas de la red; el número de entradas; y el número de capas ocultas y las neuronas de cada capa.

Arquitectura de la red

La red se compone de una serie de objetos que implementan la interfaz INeurona interconectados entre sí mediante referencias. Durante la clasificación de una entrada, cada elemento de la red colecciona los datos de todos los objetos conectados a su entrada. Cuando tiene todas las entradas de la capa anterior, calcula la salida y los pasa a todos los objetos conectados a su salida. La red funciona de forma similar pero en sentido inverso en la retropropagación. De esta forma, el funcionamiento de la red consiste en una sucesión de llamadas de los objetos INeurona que forman la primera capa hacia los que forman la capa siguiente, y así sucesivamente.

Estos objetos son creados y administrados por otro de la clase RedNeuronal, que es la que hace de interfaz con el programador. Las funciones que este objeto proporciona a este último son:

- **Cálculo**: Recibe una entrada del programador, en forma de un vector de flotantes, y pasa cada elemento del vector a cada una de las entradas de la red, lo que supone el inicio de la computación de los datos por parte de la red. Cuando la computación ha acabado, el objeto RedNeuronal compila la salida de las neuronas de la última capa y las devuelve al programador en un vector de float.
- **Entrenamiento**: Como antes, recibe del programador un vector con la entrada, pero además, recibe otro vector con los datos esperados. La red computa una salida, y a partir de ella y los datos esperados, invoca la función de entrenamiento en los datos de salida. Esta función se retropropaga por las neuronas de la red hacia atrás, cambiando los pesos de las conexiones. Al acabar el proceso, se devuelve la el control al programador, sin devolver nada.
- **Otras funciones auxiliares**, como cargar y guardar los pesos de la red (para guardar los pesos de una red entrenada, por ejemplo).



Propagación y retropropagación en la red

Objetos de la red neuronal

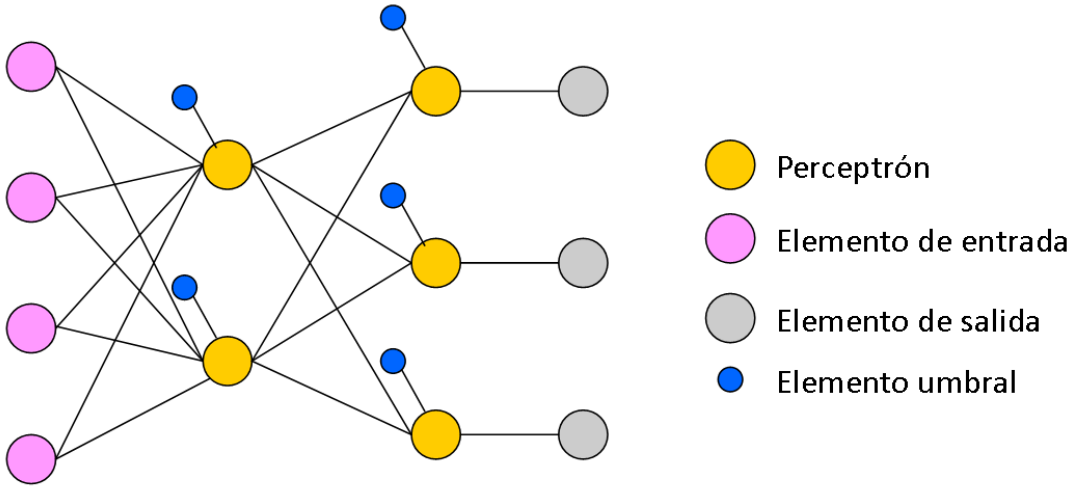
Como se dijo anteriormente, los objetos de la red neuronal están formados por objetos que implementan la interfaz INeurona. Todos estos objetos tienen cuatro funciones, que se enuncian en pseudocódigo a continuación:

- calcular (dato): clasificación
- retropropagar(parámetro de retropropagación)
- conectarEntrante (INeurona)
- conectarSaliente (INeurona)

Las dos primeras funciones son llamadas de forma recursiva de unos elementos de la red a los siguientes o precedentres, respectivamente. Las dos últimas son usadas por el objeto RedNeuronal para crear las conexiones entre los objetos INeurona.

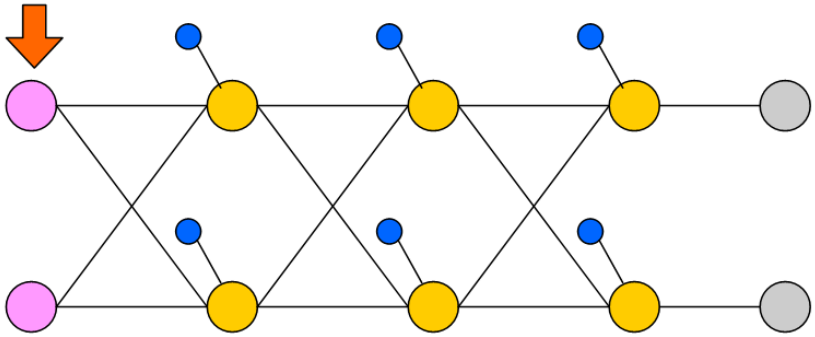
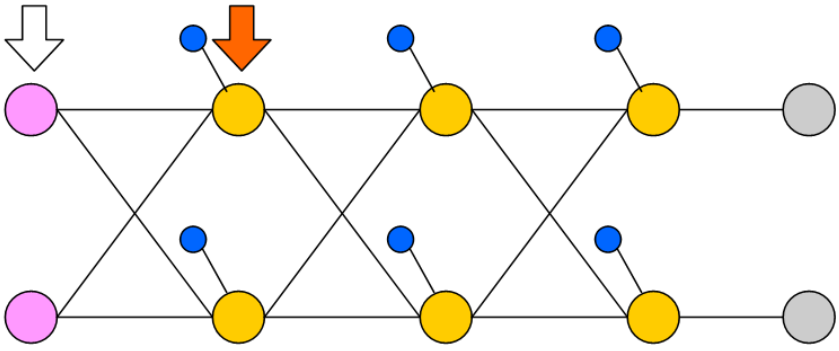
Existen cuatro tipos de objetos INeurona.

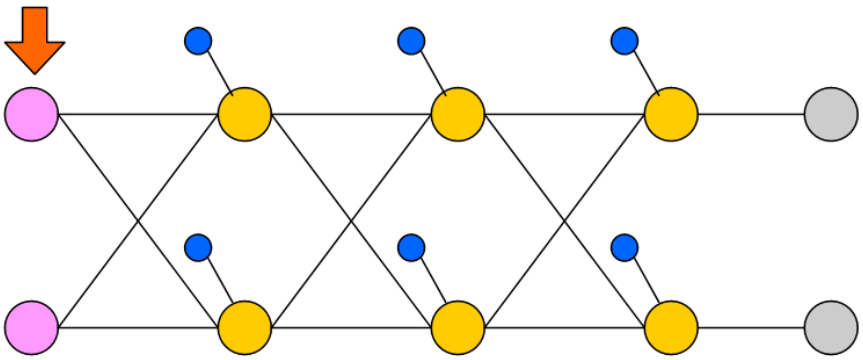
- Perceptrón: Es el único elemento de la red que realiza cálculos. Cada Perceptrón tiene una serie de elementos conectados a su entrada (de tipo Perceptrón o Elemento de entrada), que invocan su función de calcular, y una serie de elementos conectados a su salida (de tipo Perceptrón o Elemento de Salida), a los que pasar los datos calculados . Además, los elementos conectados a su salida pueden invocar su función de retropropagar.
- Elemento de entrada: Los elementos de entrada reciben un dato parcial del objeto RedNeuronal y lo propaga a todas las neuronas conectadas a él.
- Elemento de salida: Los elementos de salida están conectados a las últimas neuronas de la red. Su función es almacenar la salida hasta que el cálculo acabe en toda la red y el objeto RedNeuronal retome el control para recogerla.
- Elemento umbral: El elemento umbral es una neurona que simula la entrada con valor -1. Cada neurona de cálculo de la red (Perceptrón) tiene un elemento umbral conectado a su entrada.



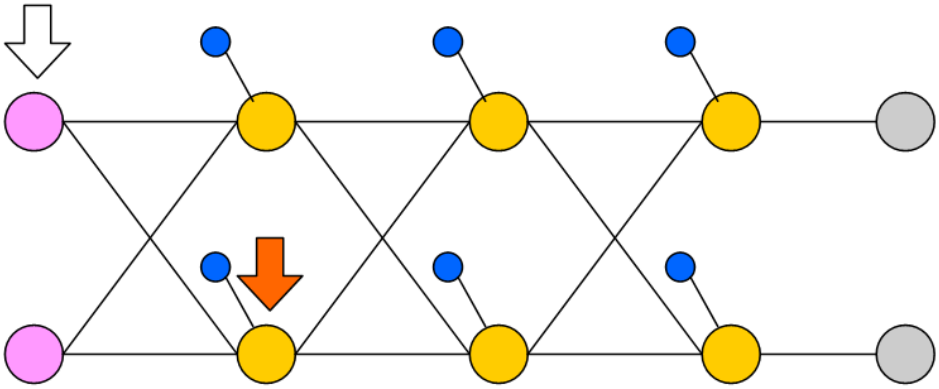
Una traza

Para ilustrar de forma gráfica el funcionamiento de la red, la siguiente tabla muestra una traza del algoritmo de cálculo para una red compuesta por dos entradas, dos capas ocultas de dos neuronas cada una, y dos salidas.

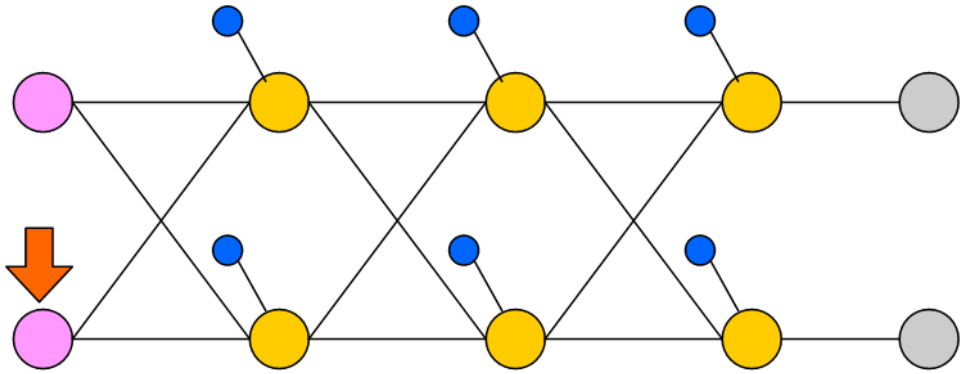
En el primer paso, el objeto RedNeuronal comienza a recorrer los elementos de entrada pasando a cada uno un valor de la entrada: 
Al tener el valor de entrada, el elemento de entrada comienza a recorrer todas las neuronas conectadas al mismo, pasándole el mismo valor que le ha pasado el objeto RedNeuronal. 
El primer perceptrón de la primera capa recibe este objeto, pero como no tiene todos los valores de sus elementos entrantes, devuelve el control al elemento de entrada:



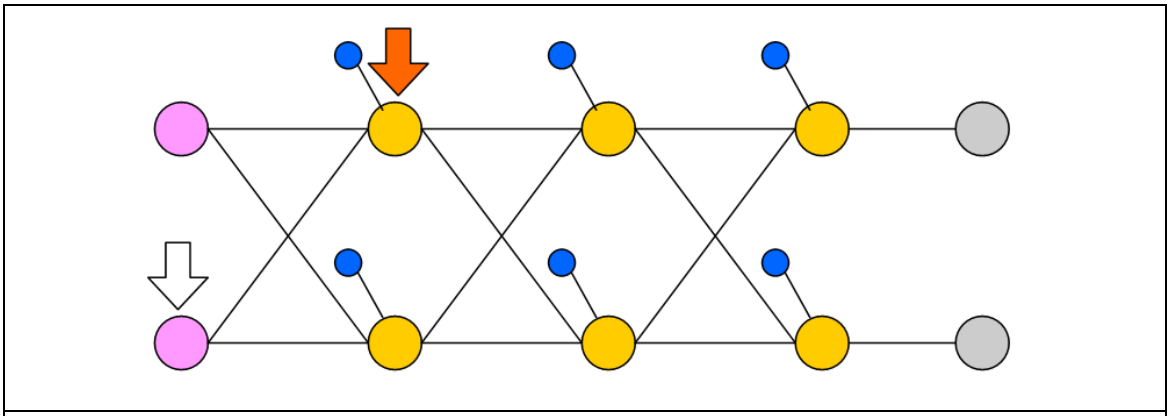
El elemento de entrada continúa con su recorrido de sus elementos salientes. En este caso, para el dato de entrada al segundo perceptrón de la primera capa:



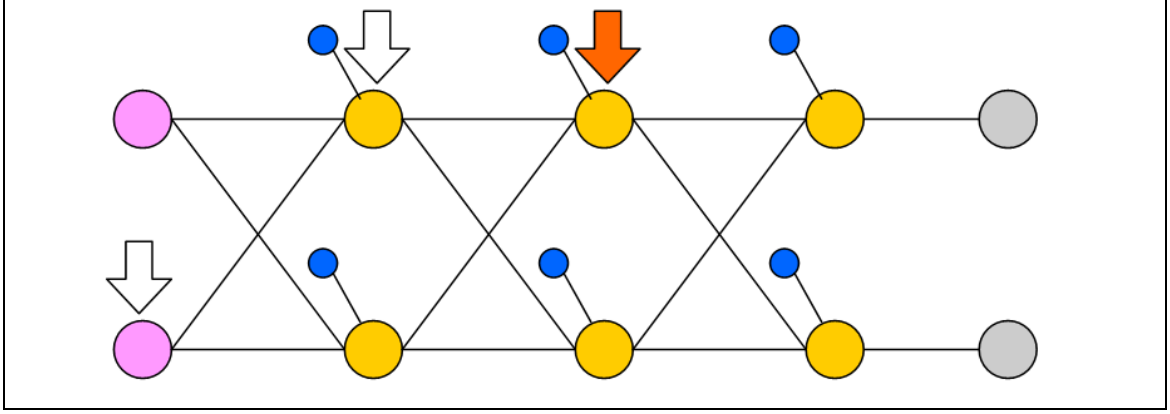
Al igual que antes, este perceptrón no puede realizar los cálculos sin todas sus entradas, así que devuelve el control al elemento entrante. Como este no tiene más conexiones salientes a los que pasar su valor, devuelve el control al objeto RedNeuronal, que le pasa el siguiente valor al siguiente objeto de entrada:



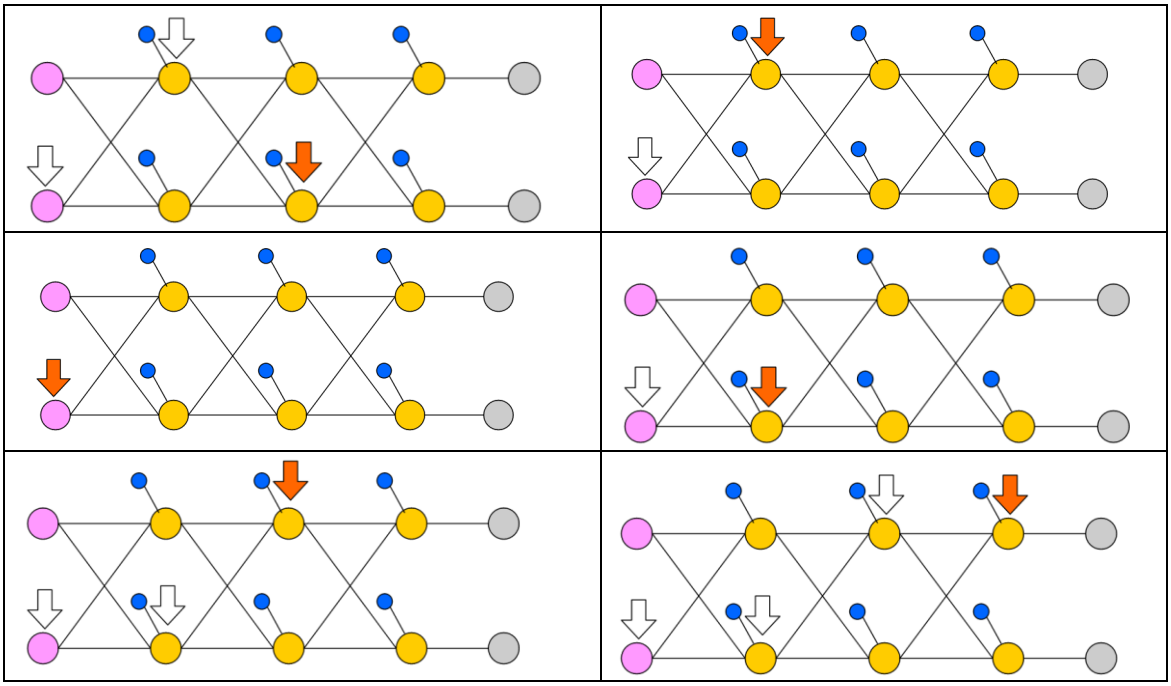
Al igual que antes, se pasa el dato al primer perceptrón:

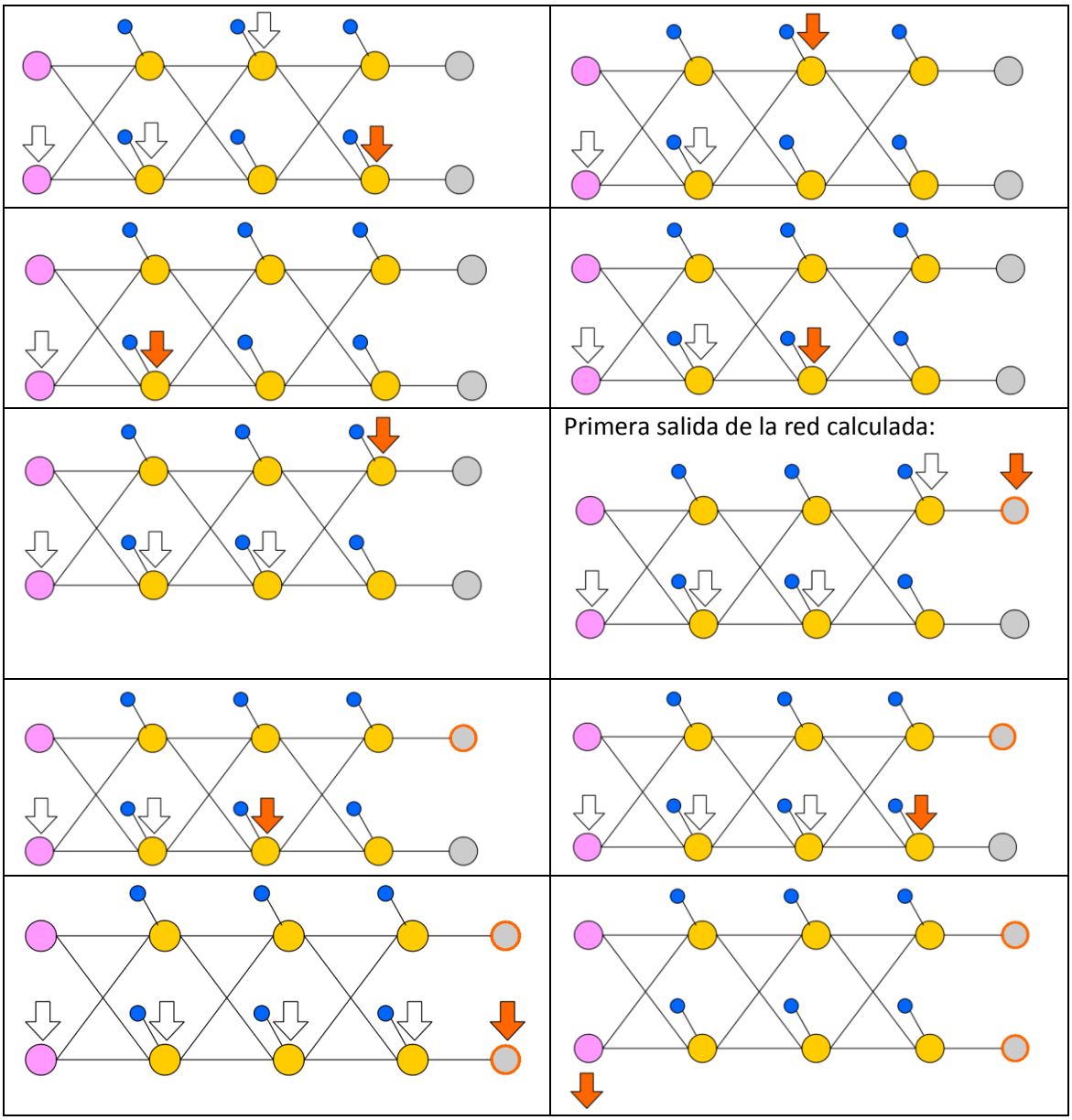


En esta ocasión, este perceptrón tiene ya todos los datos de todas las neuronas entrantes, así que puede calcular su valor de salida, y pasársela a todos los perceptrones conectados a él:



El resto de la traza se ilustra sin texto en una tabla. Léase esta de izquierda a derecha y de arriba abajo.





Tras el último paso el control vuelve al objeto RedNeuronal, que recoge las salida de los elementos de salida, la compone en un vector, y lo devuelve al usuario.

La traza del algoritmo de retropropagación es muy similar, pero en sentido inverso.

Algoritmos

Aunque el código está comentado, en esta sección vamos a describir el algoritmo de cálculo y retropropagación de los objetos Perceptrón, para facilitar su comprensión.

Cálculo

1. Entrada: Un valor float de una neurona entrante y una referencia a ésta.
2. Se guarda este valor en un diccionario de valores entrantes, cuya clave es una referencia a dicha neurona. Se guarda también en otro diccionario similar, para el algoritmo de retropropagación.
3. Si todas las neuronas entrantes han pasado ya su valor:

a. Sumar , por cada neurona entrante **n**, el valor que esta pasó por parámetros multiplicado por el peso de la conexión de este objeto con **n**.
b. Usar la función sigmoide para calcular el valor de la suma anterior. Esto es el valor de salida, que guardamos en un atributo, para el algoritmo de retropropagación.
c. Recorrer todas las neuronas de salida, invocando su función calcular, y pasando como parámetro el valor de salida y una referencia a este objeto.
d. Hacer una copia del diccionario de valores entrantes, para conservar los valores de entrada para el algoritmo de retroprogramación.
e. Vaciar de valores el diccionario de valores entrantes (ponerlos a null), y reiniciar el contador de neuronas entrantes que nos han pasado un valor de entrada.

4. Se acaba la función y se devuelve el control a la neurona que la invocó.

Retropropagación

1. Entrada: Un valor float de una neurona saliente, y una referencia a esta.
2. Se guarda este valor en un diccionario de valores procedentes de neuronas salientes, cuya clave es una referencia a dicha neurona.
3. Si todas las neuronas salientes han pasado ya su valor:
    a. Sumar , por cada neurona conectada a la salida, el valor que esta pasó por parámetros.
    b. Hacer deltaMinúscula igual al valor de salida de este objeto, v, calculado durante el algoritmo de cálculo anterior, multiplicado por (1-v), multiplicado por el sumatorio del paso anterior.
    c. Recorrer todas las neuronas entrantes, invocando su función retropropagar, y pasándole *el resultado de multiplicar deltaMinúscula* ***por el peso de la conexión*** *entre este objeto y la neurona entrante*, además de una referencia a este objeto. *Nótese cómo la multiplicación por los pesos lo hacemos en este paso, de manera que esta multiplicación no hay que hacerla en el paso 3.a.* Como es una llamada síncrona, el algoritmo espera a que la última neurona entrante acabe su función de retropropagación para seguir.
    d. Por cada neurona entrante, **ni**:
        i. Hacer **diferencial** igual a la suma de:
            1. El factor de aprendizaje, multiplicado por deltaMinúscula, multiplicado por la última entrada procedente de ni (esta entrada se guardó en el paso 3.d. del algoritmo de cálculo).
            2. El factor de momento, por el diferencial obtenido en una retropropagación anterior.
        ii. Guardar este diferencial, asociado a la neurona ni, para calcular el momento en una posible retroprogación posterior.
        iii. Sumar diferencial al peso de la conexión entre ni y este objeto.
    e. Vaciar de valores el diccionario de valores procedentes de neuronas salientes (ponerlos a null), y reiniciar el contador de neuronas salientes que nos han pasado un valor de retropropagación.

4. Se acaba la función y se devuelve el control a la neurona que la invocó.

4. Pruebas y resultados experimentales

Metodología

Hemos hecho dos experimentos, uno para aprender parte del alfabeto del lenguaje de signos, y otro para reconocer parte del alfabeto del lenguaje de señales con banderas. Para cada experimento, hemos hecho dos bloques de pruebas: uno para averiguar cuál puede ser la mejor configuración, y otro para averiguar cuál puede ser la mejor estructura de la red.

Dado que cada entrenamiento comienza con pesos distintos, y el orden de uso del conjunto de entrenamiento es aleatorio, haremos cuatro ejecuciones del entrenamiento por cada configuración en cada bloque.

Primer bloque de pruebas: parámetros

Para este bloque de prueba, la configuración de la red tendrá una parte fija para todas las pruebas y otra variable, que se reflejará en las tablas de resultados. La parte fija de la configuración es la siguiente:

- Pesos iniciales mínimos y máximos, fijados a -0.5 y 0.5 respectivamente. En general, estos valores dan buen resultado, a la vez que producen un rango amplio de computaciones.
- Iteraciones máximas, fijadas a 100. Dado que en las pruebas previas hemos comprobado que 100 iteraciones, con la configuración adecuada, es más que suficiente para conseguir una cobertura total del conjunto de parada, hemos decidido limitar las iteraciones a este número, para no eternizar las pruebas.
- Estructura de la red: Una capa oculta de seis neuronas. Aunque hemos detectado en las pruebas previas que con más neuronas conseguimos, en general, mejores resultados o con menos iteraciones, hemos comprobado también que se pueden conseguir una cobertura total del conjunto de parada con una capa oculta de seis neuronas.

Segundo bloque de pruebas: estructura de la red

En este bloque de prueba, elegiremos una configuración de la red que haya dado buenos resultados en el bloque anterior, y probaremos con varias configuraciones de red. Mediremos el tiempo aproximado de cada iteración de entrenamiento, para poder comparar diversas estructuras.

Primer experimento: Reconocimiento de alfabeto de símbolos

Reconocimiento de 8 letras. 177 fotografías en el conjunto de entrenamiento, 16 en el de parada.

Primer bloque: parámetros

Factor de aprendizaje	Factor de momento	Iteraciones	Porcentaje de ajuste conjunto de parada

0.1	**0**	100	100	100	100	68	37	50	75
	0.1	100	100	100	100	25	12	62	37
	0.3	100	100	100	100	75	87	25	75
	0.5	100	100	100	100	50	75	87	62
0.3	**0**	100	75	100	100	87	100	62	81
	0.1	100	100	100	100	75	87	43	50
	0.3	**88**	**39**	100	100	**100**	**100**	87	75
	0.5	100	100	100	100	75	37	75	75
0.5	**0**	76	100	100	100	100	87	75	50
	0.1	100	100	100	100	62	**100**	87	68
	0.3	100	100	100	100	62	75	75	81
	0.5	100	100	100	**93**	50	75	87	**100**
0.8	**0**	100	**71**	100	100	50	**100**	75	87
	0.1	100	100	100	100	75	25	75	75
	0.3	**11**	100	100	100	**100**	0	50	75
	0.5	100	100	100	100	0	0	0	12
1	**0**	**73**	100	100	100	**100**	25	37	0
	0.1	100	100	100	100	25	87	12	37
	0.3	100	100	100	100	18	25	25	12
	0.5	100	100	100	100	12	0	0	0

Vemos cómo los mejores resultados se dan con factores de aprendizaje alrededor de 0.3, aunque los resultados de 0.5 tampoco son malos. Un factor de aprendizaje demasiado pequeño puede enlentecer demasiado el proceso de aprendizaje, y un factor de aprendizaje de 1, sobre todo con momento, puede hacer que la red no converja a ninguna solución, como era de esperar. De hecho, una observación detenida de una computación se puede ver como con factor de aprendizaje uno, el porcentaje de aciertos puede variar significativamente entre cada iteración de entrenamiento.

Un factor de momento pequeño puede ayudar a evitar mínimos locales en algunas computaciones. Una observación atenta puede confirmar como con un factor momento considerable (~0.5), el porcentaje de aciertos a veces fluctúa (sube y baja) durante varias iteraciones hasta fijarse en un valor concreto. Un momento elevado puede hacer que la red no converja a un mínimo cuando lo combinamos con un factor de aprendizaje elevado (e.g.: 0.8).

Segundo bloque: estructura de la red

En este bloque vamos a probar diferentes estructuras de red tomando como factor de aprendizaje 0.5 y factor de momento 0.3. En configuraciones con muchas neuronas vamos a limitar el número de iteraciones a 50 o 30.

Composición de la red (capas ocultas)	**Tiempo aproximado por iteración**	**Iteraciones**				**Porcentaje**			
2	1s	100	100	100	100	12	12	18	0

6	2s	100	100	100	100	62	87	87	43
12	5s	23	34	26	33	**100**	**100**	**100**	87
20	12s	20	12	70	21	**100**	**100**	80.7	100
50	30s	30	16	12	19	87	**100**	**100**	**100**
12;4	4s	100	100	100	100	75	75	56	75

Vemos cómo aumentando el número de neuronas conseguimos mejores resultados con un tiempo similar. Sin embargo, existe riesgo de sobreajuste.

En el caso de la segunda computación con 20 neuronas concluimos que estábamos en un mínimo local, ya que desde la iteración 14 el valor era el mismo, y abortamos la ejecución.

No parece que obtengamos una gran ventaja usando más de una capa oculta. En el ejemplo con doce neuronas en la primera capa y cuatro en la segunda, parece no poder ajustarse al conjunto de entrenamiento. Otras combinaciones que hemos hecho con anterioridad tampoco han resultado interesantes.

Segundo experimento: Reconocimiento de señales de banderas

Reconocimiento de cinco letras. 118 fotografías en el conjunto de entrenamiento, 10 en el de parada. En estas fotografías no está eliminado el fondo, y aparecen personas con ropa normal.

Primer bloque de pruebas: parámetros

Factor de aprendizaje	**Factor de momento**	**Iteraciones**				**Porcentaje**			
0.1	0	100	100	100	100	20	40	60	60
	0.3	100	100	100	100	50	80	20	40
0.3	0	**63**	**56**	**93**	**85**	**100**	**100**	**100**	**100**
	0.3	**73**	100	100	100	**100**	50	60	20
0.5	0	**57**	100	**38**	**48**	**100**	20	**100**	**100**
	0.3	100	**34**	**43**	**27**	80	**100**	**100**	**100**
0.8	0	**78**	**64**	**55**	**42**	**100**	**100**	**100**	**100**
	0.3	100	100	**61**	100	40	20	**100**	40
1	0	**51**	**40**	**78**	**10**	**100**	**100**	**100**	**100**
	0.3	100	100	100	100	20	60	0	40

Es interesante observar cómo en este segundo experimento obtenemos resultados mucho mejores, incluso cuando hemos tenido menos cuidado en conseguir fotos especialmente preparadas para el experimento (sin tratar el fondo, personas con ropa normal…). Estos resultados deben deberse a que se realiza la clasificación de las fotos en un conjunto menor (cinco fotos frente a ocho). En otros experimentos previos, cuando tratábamos de reconocer no ocho, sino cinco letras del alfabeto de signos, también obteníamos mejores resultados.

Segundo bloque de pruebas: Configuración de la red

Usamos los mismos valores que en el experimento anterior, factor de aprendizaje de 0.5 y factor de momento de 0.3.

Composición de la red (capas ocultas)	**Tiempo aproximado por iteración**	**Iteraciones**				**Porcentaje**			
2	1/2s	100	100	100	100	40	40	20	80
6	2s	**47**	**75**	**65**	100	**100**	**100**	**100**	40
12	4s	**30**	**29**	**31**	**38**	**100**	**100**	**100**	**100**
20	7s	**20**	**30**	**26**	**23**	**100**	**100**	**100**	**100**
50	15s	**21**	**19**	**13**	**10**	**100**	**100**	**100**	**100**
12;4	4s	**59**	100	100	**81**	**100**	60	80	**100**

En este experimento podemos ver aun más claro cómo podemos conseguir ajustes precisos en un tiempo similar aumentando el número de neuronas, con aproximadamente la mitad de iteraciones.

La configuración de doble capa oculta ha tenido mejor resultado en este experimento que en el anterior, seguramente por el menor número de clasificaciones a realizar. Aun así, obtenemos mejores resultados con una sola capa oculta.