Algoritmo de retropropagación

Hemos decidido dos algoritmos distintos de feedforward, el primero es `RedNeuronal::train`

```
// Por cada epoca de entrenamiento
for (int epocas = 0; epocas < max_epocas; epocas++)
{
    // Por cada vector de entrenamiento
    for (int i = 0; i < num_rows_train; i++)
    {
        // Feedforward
        feedforward(entrada_entrenamiento[i]);

        // Calculamos deltas y actualizamos los pesos
        aprender(salida_entrenamiento[i]);
    }
}
```

Y el segundo es `RedNeuronal::train2`

```
// Por cada epoca de entrenamiento
for (int epocas = 0; epocas < max_epocas; epocas++)
{
    // Por cada vector de entrenamiento
    for (int i = 0; i < num_rows_train; i++)
    {
        // Feedforward
        feedforward(entrada_entrenamiento[i]);

        // Calculamos los deltas y los guardamos
        aprender2(salida_entrenamiento[i]);
    }

    // Actualizamos los pesos de la red haciendo la media de los deltas
    gradient_descent();
}
```

La funcion de feedforward es muy simple, hacemos el siguiente calculo:

```
// Guardamos la entrada en la capa de entrada
capa_entrada = entrada;

// Hacemos (capa_entrada * entrada-oculta_peso + entrada-oculta_sesgo)
capa_oculta_tmp = add_tensors(dot_product(capa_entrada, eo_peso), eo_sesgo);

// Calculamos la sigmoidal (ya sea bipolar o binaria)
capa_oculta = sigmoide(capa_oculta_tmp, bi);

// Hacemos (capa_oculta * oculta-salida_peso + oculta-salida_sesgo)
capa_salida_tmp = add_tensors(dot_product(capa_oculta, os_peso), os_sesgo);

// Calculamos la sigmoidal (ya sea bipolar o binaria)
capa_salida = sigmoide(capa_salida_tmp, bi);
```

La funcion de aprendizaje sigue los siguientes pasos:

1. Calculamos delta_salida.

```
delta_salida = (salida_esperada - capa_salida) * f'(capa_salida_tmp)
```

2. Calculamos delta_w.

```
delta_w[k][j] = tasa_aprendizaje * delta_salida[k] * capa_oculta[j]
```

3. Calculamos delta_in.

```
delta_in[j] = delta_salida[k] * os_peso[k][j]
```

4. Calculamos delta_oculta.

```
delta_oculta[j] = delta_in[j] * f'(capa_oculta_tmp[j], bi)
```

5. Calculamos delta_v.

```
delta_v[j][i] = tasa_aprendizaje * delta_oculta[j] * capa_entrada[i]
```

6. Dependiendo de la funcion podemos hacer dos cosas.
   - `RedNeuronal::aprender`: actualizamos los pesos de la red.
   - `RedNeuronal::aprender2`: guardamos `delta_v`, `delta_w`, `delta_sv`, `delta_sw` para actualizar luego.

Resultados con problemas reales

Problema real 1

Fichero Datos	Modo Lectura	Capa oculta	Tasa aprendizaje	Bipolar Binaria	Algoritmo	Max iter	Error Medio	TP	FP	FN	TN	Acierto (%)
data/problema_real1.txt	2	4	0.2	0	0	1000	0.0203424	684	21	15	678	97.4249
data/problema_real1.txt	2	6	0.4	1	0	1000	0.0112019	691	8	8	691	98.8555
data/problema_real1.txt	**2**	**6**	**0.4**	**1**	**0**	**5000**	**0.00431206**	**696**	**3**	**3**	**696**	**99.5708**

Problema real 2

Fichero Datos	Modo Lectura	Capa oculta	Tasa aprendizaje	Bipolar Binaria	Algoritmo	Max iter	Error Medio	TP	FP	FN	TN	Acierto (%)
data/problema_real2.txt	2	4	0.2	0	0	1000	0.142212	449	118	110	441	79.6064
data/problema_real2.txt	2	2	0.1	0	0	1000	0.149423	458	135	101	424	78.8909
data/problema_real2.txt	2	6	0.1	1	0	1000	0.135822	454	104	105	455	81.2165
data/problema_real2.txt	2	8	0.4	1	0	1000	0.114294	467	92	92	467	83.542
data/problema_real2.txt	2	6	0.2	1	0	1000	0.130092	467	93	92	466	83.3631
data/problema_real2.txt	2	6	0.3	1	0	1000	0.122848	465	94	94	465	83.1843
data/problema_real2.txt	2	6	0.4	1	0	1000	0.113762	479	80	80	479	85.6887
data/problema_real2.txt	2	6	0.5	1	0	1000	0.118852	470	89	89	470	84.0787
data/problema_real2.txt	2	7	0.4	1	0	1000	0.116255	472	87	87	472	84.4365
data/problema_real2.txt	2	6	0.4	0	0	5000	0.136935	482	147	77	412	79.9642
data/problema_real2.txt	**2**	**6**	**0.4**	**1**	**0**	**5000**	**0.103789**	**486**	**73**	**73**	**486**	**86.941**

Problema real 3

Fichero Datos	Modo Lectura	Capa oculta	Tasa aprendizaje	Bipolar Binaria	Algoritmo	Max iter	Error Medio	TP	FP	FN	TN	Acierto (%)
data/problema_real3.txt	2	2	0.4	1	0	5000	8.05E-06	141	9	9	291	96
data/problema_real3.txt	2	3	0.4	1	0	5000	8.13E-06	140	10	10	290	95.3333
data/problema_real3.txt	2	4	0.4	1	0	5000	8.38E-06	148	2	2	298	98.6667
data/problema_real3.txt	**2**	**6**	**0.4**	**1**	**0**	**5000**	**6.46E-06**	**149**	**1**	**1**	**299**	**99.3333**

Problema real 5

Fichero Datos	Modo Lectura	Capa oculta	Tasa aprendizaje	Bipolar Binaria	Algoritmo	Max iter	Error Medio	TP	FP	FN	TN	Acierto (%)
data/problema_real5.txt	2	6	0.4	1	0	5000	0.00116414	434	1	1	434	99.7701
data/problema_real5.txt	**2**	**6**	**0.2**	**1**	**0**	**5000**	**2.27E-05**	**435**	**0**	**0**	**435**	**100**

Comentario de los resultados

Como se puede observar en la tabla, todos los problemas excepto el segundo nos dan unos resultados excelentes (>99%).

Hemos encontrado que la siguiente es la arquitectura que mejor funcionaba:

- Tasa de aprendizaje: [0.2 - 0.4]
- Neuronas en la capa oculta: 6
- Iteraciones: ~5000
- Funcion: Binaria

Esta arquitectura se ha descubierto poco a poco a lo largo de la experimentación con el problema real 2 (que ha sido el más costoso de optimizar).

También hemos descubierto que el uso de menos neuronas, no reduce demasiado los resultados (~2% menos de acierto).

Problemas reales y normalización

Problema 4. Intento sin normalización

Fichero Datos	Modo Lectura	Capa oculta	Tasa aprendizaje	Bipolar Binaria	Algoritmo	Max iter	Error Medio	TP	FP	FN	TN	Acierto (%)
data/problema_real4.txt	2	6	0.1	1	1	100	0.231593	455	240	244	459	65.3791
data/problema_real4.txt	2	12	0.4	1	1	1000	0.212386	471	230	228	469	67.2389
data/problema_real4.txt	2	20	0.4	1	1	1000	0.212206	472	236	227	463	66.8097
data/problema_real4.txt	**2**	**6**	**0.4**	**1**	**1**	**1000**	**0.190318**	**520**	**179**	**179**	**520**	**74.392**

Se puede observar que no pasamos del ~75% en ninguno de los intentos

Problema 4. Intento con normalización

Hemos usado la siguiente fórmula (siendo max el valor más alto de la columna):

```
valor = (valor / max) * 2 - 1;
```

Y la hemos aplicado a todos los valores de entrada.

Fichero Datos	Modo Lectura	Capa oculta	Tasa aprendizaje	Bipolar Binaria	Algoritmo	Max iter	Error Medio	TP	FP	FN	TN	Acierto(%)
data/problema_real4.txt	2	6	0.1	1	1	100	0.214154	460	241	239	458	65.6652
data/problema_real4.txt	2	6	0.1	1	1	1000	0.0477582	669	32	30	667	95.5651
data/problema_real4.txt	2	6	0.1	1	1	5000	0.0252849	679	22	20	677	96.9957
data/problema_real4.txt	2	20	0.1	1	1	5000	0.024576	679	22	20	677	96.9957
data/problema_real4.txt	2	10	0.1	1	1	5000	0.0253082	678	20	21	679	96.9957
data/problema_real4.txt	2	14	0.1	1	1	5000	0.0247694	677	20	22	679	96.9957
data/problema_real4.txt	**2**	**12**	**0.1**	**1**	**1**	**5000**	**0.0250298**	**679**	**20**	**20**	**679**	**97.1388**

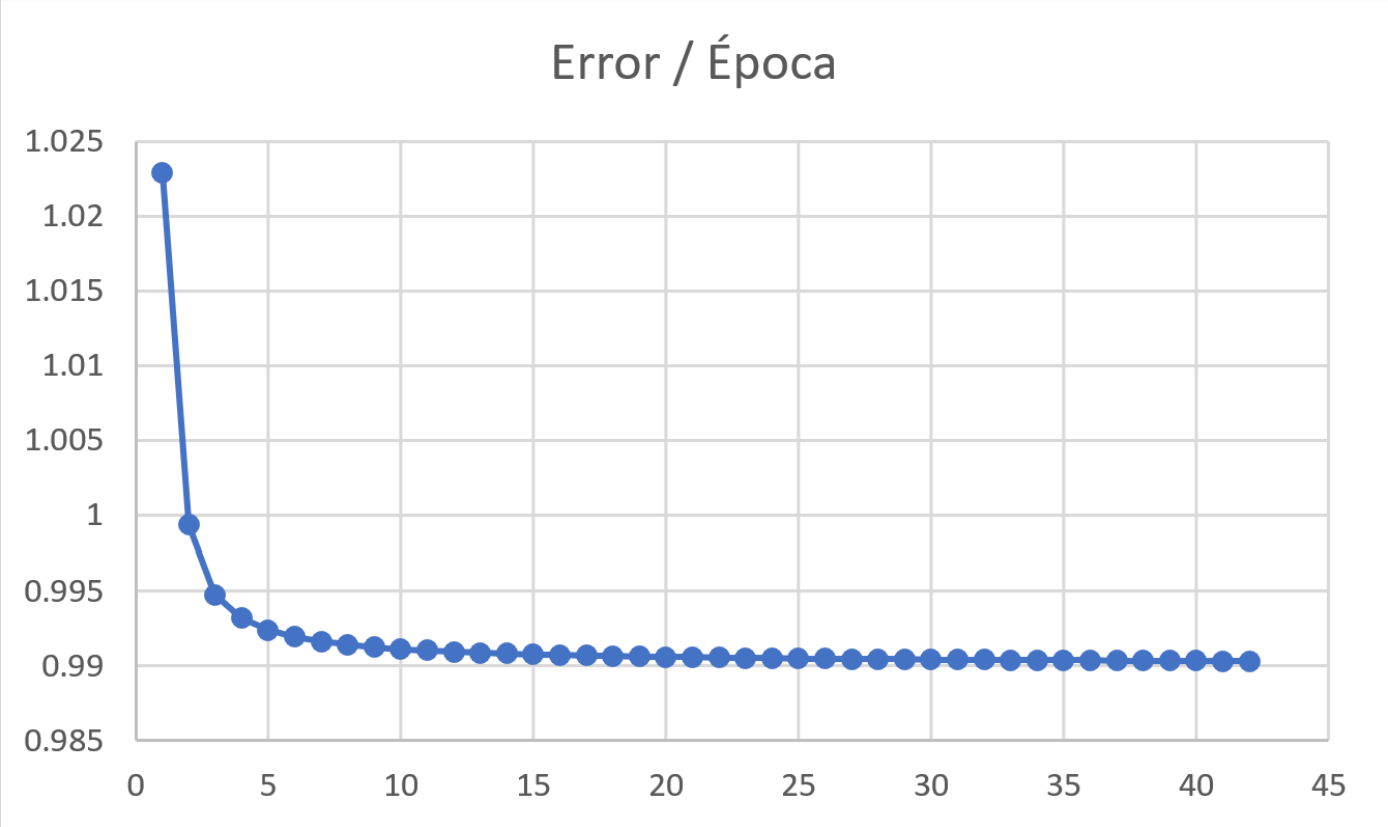
Se puede ver que el salto ha sido extremadamente grande: 74% -> 97%

Problema 6

En el problema 6, debido a la longitud de las pruebas, no hemos probado sin normalización. Hemos asumido que la normalización ha quedado demostrada como beneficiosa en el problema 4.

Fichero Datos	Modo Lectura	Capa oculta	Tasa aprendizaje	Bipolar/Binaria	Algoritmo	Maxiter	Error Medio	TP	FP	FN	TN	Acierto (%)
data/problema_real6.txt	2	12	0.1	1	1	100	0.247481	8257	6722	6723	8258	55.1202
data/problema_real6.txt	1	6	0.1	1	1	1000	0.249641	1591	1405	1405	1591	53.1041
data/problema_real6.txt	1	6	0.4	1	1	1000	0.249669	1591	1405	1405	1591	53.1041
data/problema_real6.txt	1	12	0.1	1	1	1000	0.249663	1590	1405	1406	1591	53.0708
data/problema_real6.txt	1	12	0.1	1	1	1000	0.24969	1591	1405	1405	1591	53.1041
data/problema_real6.txt	1	12	0.4	1	1	1000	0.249666	1591	1405	1405	1591	53.1041
data/problema_real6.txt	1	12	0.1	1	0	100	0.250137	1591	1405	1405	1591	53.1041
data/problema_real6.txt	1	20	0.15	1	1	1000	0.249716	1591	1406	1405	1590	53.0708
data/problema_real6.txt	1	20	0.1	1	1	1000	0.249619	1591	1406	1405	1590	53.0708
data/problema_real6.txt	1	20	0.05	1	1	1000	0.249686	1590	1405	1406	1591	53.0708
data/problema_real6.txt	1	20	0.15	1	0	1000	0.252907	1610	1386	1386	1610	53.7383
data/problema_real6.txt	1	20	0.1	1	0	1000	0.25214	1606	1390	1390	1606	53.6048
data/problema_real6.txt	1	32	0.1	1	1	1000	0.249712	1591	1406	1405	1590	53.0708
data/problema_real6.txt	1	12	0.1	1	0	1000	0.251691	1607	1389	1389	1607	53.6382
data/problema_real6.txt	1	12	0.4	1	0	1000	0.257143	1594	1402	1402	1594	53.2043
data/problema_real6.txt	1	6	0.1	1	0	1000	0.252673	1598	1398	1398	1598	53.3378
data/problema_real6.txt	1	6	0.4	1	0	1000	0.256928	1595	1401	1401	1595	53.2377
data/problema_real6.txt	**1**	**20**	**0.05**	**1**	**0**	**1000**	**0.25038**	**1620**	**1376**	**1376**	**1620**	**54.0721**

Al ver que no encontrábamos una arquitectura de red que nos diera buenos resultados hemos creado la siguiente gráfica para comprobar si era un problema con el número de iteraciones:



Queda claro entonces que no es un problema con el número de épocas que empleamos en el entrenamiento. También hemos intentado un número alto de neuronas en la capa oculta (64 y 128 neuronas) pero los resultados han sido los mismos.