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

Fichero Datos	Modo Lectura	Capa oculta	Tasa aprendizaje	Bipolar/Binaria	Algoritmo	Maxiter	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
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
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
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