CC50Q
Ejercicio 10
Comparando los errores *batch* y *on-line*

Francisca Muñoz

9 de noviembre de 2005

1. Introducción

El objetivo de este ejercicio es comparar la evolución del error aplicando distintos criterios de entrenamiento a una neurona simple, que utiliza método del descenso del gradiente. Las variantes a aplicar son *entrenamiento batch* y *entrenamiento on-line*. El *entrenamiento batch* consiste en evaluar el error total sobre todos los ejemplos, y en base a esto, actualizar los pesos de la neurona. El *entrenamiento on-line* en tanto, actualiza los pesos de la neurona cada vez que procesa un nuevo ejemplo.

2. Algoritmos

Introducción Los algoritmos fueron implementados en *octave*, un software similar a *matlab* pero de desarrollo abierto.

El programa `principal.m` hace un loop sobre los distintos conjuntos de ejemplos y ejecuta sobre ellos los dos métodos de entrenamiento.

Los programas `entrena_batch.m` y `entrena_online.m` son muy similares. Ambos utilizan las mismas funciones de inicialización, combinación lineal de las entradas, de

activación y de error para ir ajustando los parámetros w que separan el conjunto de forma apropiada.

La diferencia ente ambos es el momento en que se acutalizan los parámetros o pesos w, como se puede ver en las líneas 14 y 18.

```
 1 function [w, iteracion]=entrenar_batch(Datos,mu,epsilon,fid)
 2 [X, D, N, I, w] = initialize(Datos);
 3
 4 E=epsilon+1;
 5 iteracion=1;
 6 while (E > epsilon)
 7        E=0;
 8        dE_dw=zeros(I+1,1);
 9        % iteramos sobre los ejemplos
10        for i=1:1:N
11              [E,y_i] = acumula_error_i(E,X,D,w,i);
12              [dE_dw] = backpropagation_i(dE_dw,y_i,X,D,w,i);
13                % >> actualizar los pesos on-line
14                % w=w-mu*dE_dw;
15        end
16        fprintf(fid,'%d %f\n',iteracion,E);
17        % >> actualizar los pesos batch
18        w=w-mu*dE_dw;
19
20        iteracion=iteracion+1;
21 end
```

Ambos métodos iteran sobre el conjunto de ejemplos la cantidad de veces que sea necesaria hasta que el error haya disminuído hasta una cierta cota *epsilon* (en la línea 4 se hace $E = epsilon + 1$ para que se entre al menos una vez al loop). Luego para cada ejemplo i, se calcula el error de la salida de la neurona respecto a la clase a la que realmente pertenece (línea 11), y encuentra el gradiente por medio de la retropropagación (línea 12). Una vez terminada la revisión de los ejemplos, se imprime la iteración y el error en un archivo para posteriormente graficarlos, como se ve en la línea 15.

Inicialización Lo primero que se hace en el proceso de aprendizaje es preparar los datos para comenzar el proceso iterativo, e inicializar las variables necesarias.

Los datos vienen en un archivo que contiene un ejemplo por cada fila, con sus atributos o características, y la clase a la que pertenece (denominada por un 0 o un 1, ya que usaremos la sigmoide logística como función de activación).

N es la cantidad de ejemplos con que se cuenta, I es la cantidad de atributos que presentan los ejemplos, X son los atributos de los datos, D son las clases a las que pertenece cada dato, y w es el vector de pesos (mismo tamaño que la cantidad de atributos)

La matriz de datos que se lee tiene tamaño N por $(I + 1)$, y para entregar estos datos a los programas de entrenamiento, separamos los atributos de su clasificación, y agregamos una columna de unos a los atributos, que corresponde a la entrada asociada al parámetro w_0, que no modula a las entradas. (líneas 4 hasta 6).

En la línea 7 se inicializa la función aleatoria `randn`, que es con la que se calcula w, para obtener los mismos vectores de pesos para ambas corridas. (`randn(x,y)` retorna una matriz de tamaño x por y de valores seudoaleatorios, obtenidos de una distribución uniforme sobre el intervalo unitario).

```
1 function [X, D, N, I, w]=initialize(Datos)
2 N=size(Datos,1);
3 I=size(Datos,2)-1;
4 X_original=Datos(:,1:I);
5 D=Datos(:,I+1);
6 X=[ones(N,1) X_original];
7 rand('seed',5);
8 w = rand(I+1,1);
```

Cálculo del Error El primer paso en la iteración sobre los ejemplos es realizar la **combinación lineal** sobre la entrada x, multiplicándola por sus pesos asociados.

```
1 a_i=X(i,:)*w;
```

Luego, **la función de activación** asigna valores cercanos a 1 o 0 al ejemplo basado en los valores de la etapa anterior.

```
2 y_i=1/(1+exp(-a_i));
```

Para ambos conjuntos de entrenamiento, consideramos el **error cuadrático** parcial para cada ejemplo, y el error total es la suma de estos errores parciales.

```
3 d_i=D(i,1);
4 e_i=(d_i-y_i)^2;
5 E=E+e_i;
```

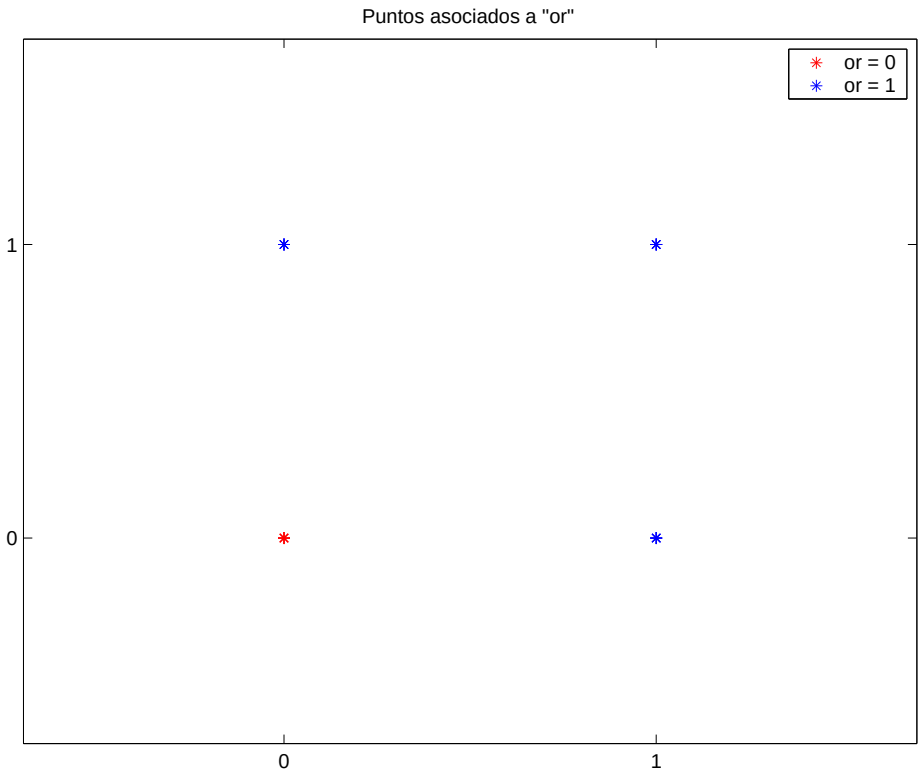
Retropropagación del Error En esta sección, obtenemos la derivada del error respecto a los pesos a través de la regla de la cadena. Vemos en la línea 1 la derivada del error respecto a y, en la línea 2 la derivada de y respecto a a (derivada de la sigmoide), en la línea 3 la derivada de a respecto a w (solo sobrevive un término por cada componente), en la línea 4 la multiplicación de los términos por regla de la cadena, y finalmente la actualización de; gradiente del error en la línea 10.

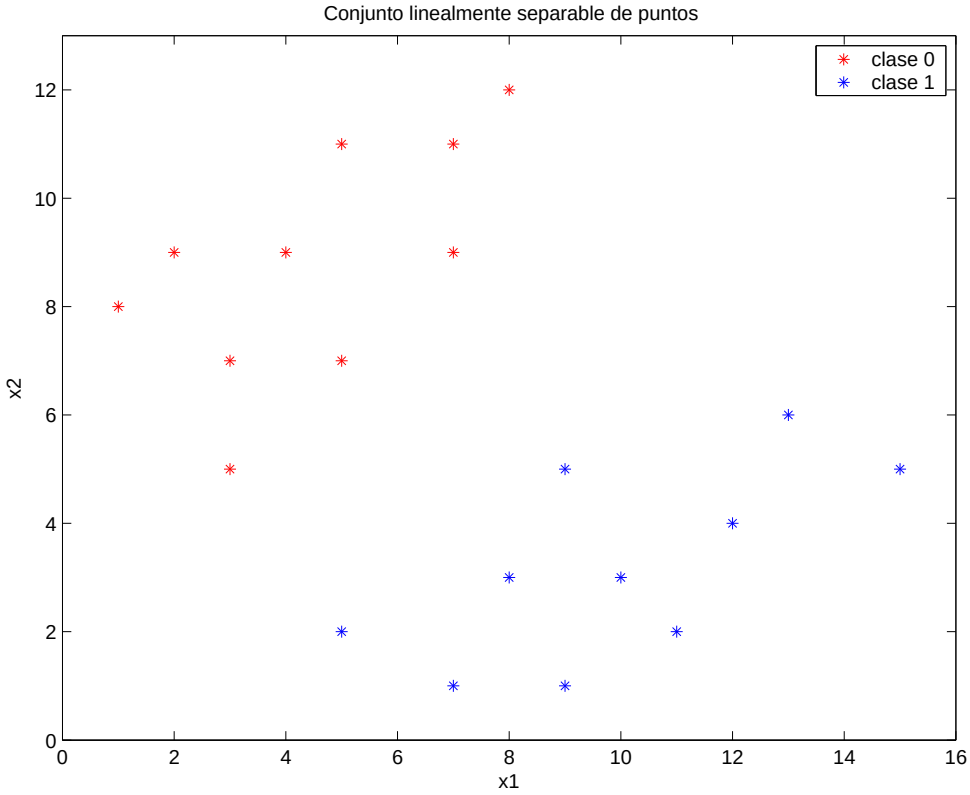
```
1 de_dy=-2*(d_i-y_i);
2 dy_da=y_i*(1-y_i);
3 da_dw=X(i,:)';
4 de_dw=de_dy*dy_da*da_dw;
5 dE_dw=dE_dw+de_dw;
```

3. Conjuntos de Entrenamiento

Para el caso de la neurona simple, para que pueda diferenciar las clases de los conjuntos de entrada, estos deben ser linealmente separables. En este caso se ha elegido el caso del *or* lógico (visto en clases), las letras ascii publicadas por Francisco Claude en ascii que pueden ser vocales o consonantes, y un conjunto de 20 puntos linealmente separables.

Aquí están los gráficos de los puntos asociados al *or* y a los puntos linealmente separables.

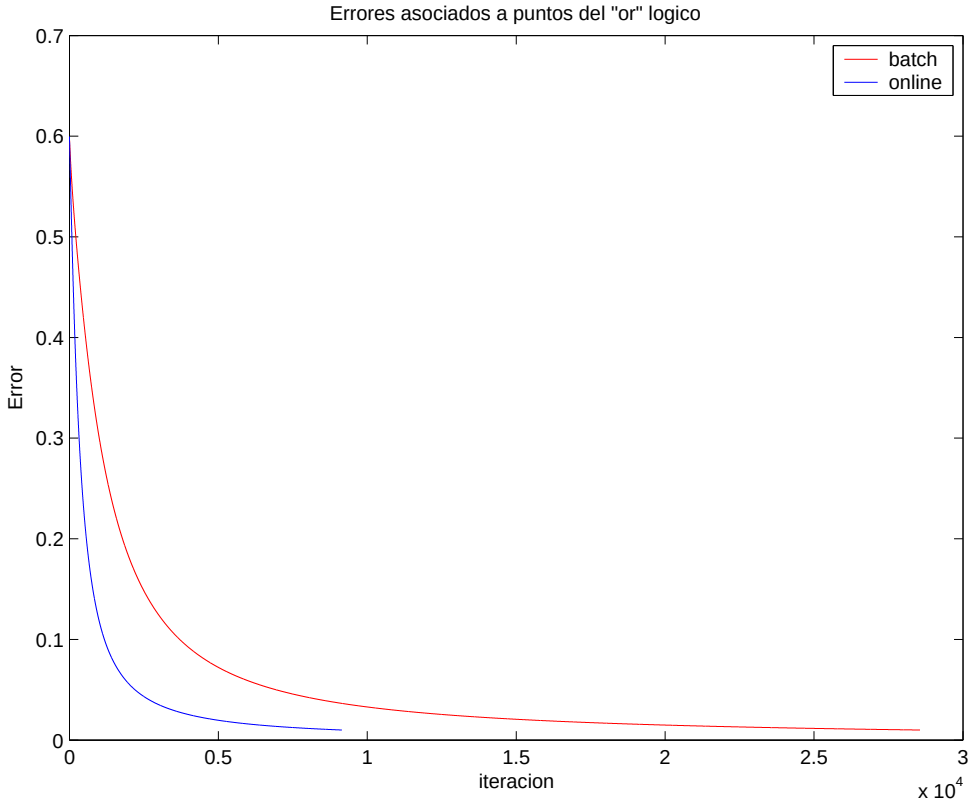


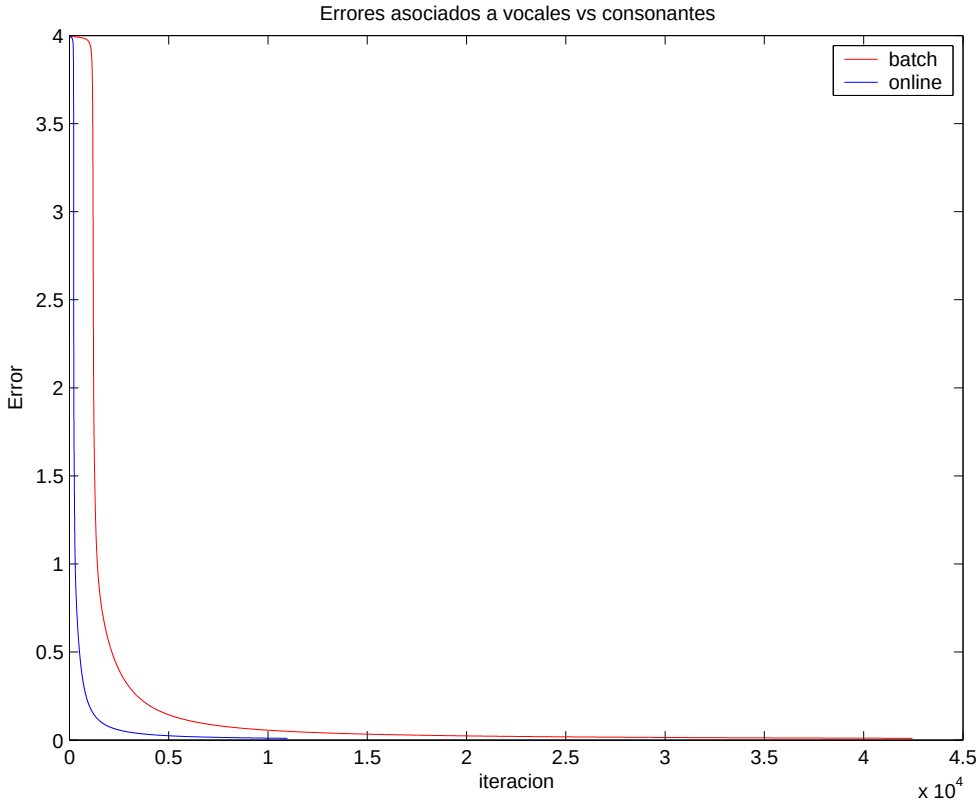


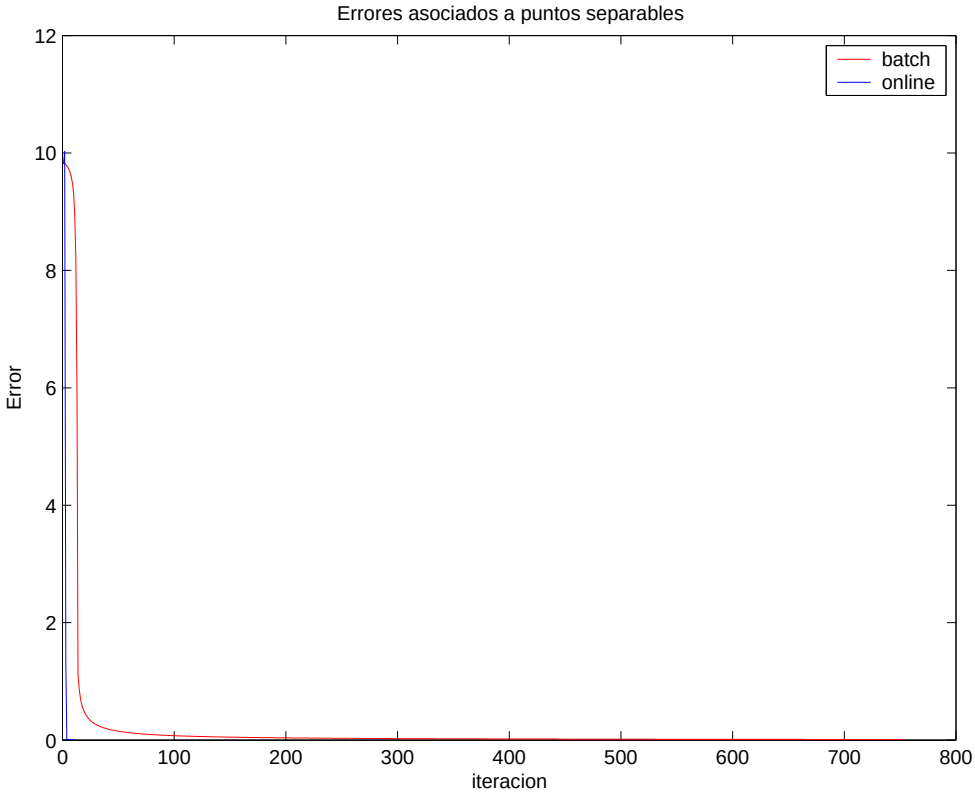
4. Experimentos

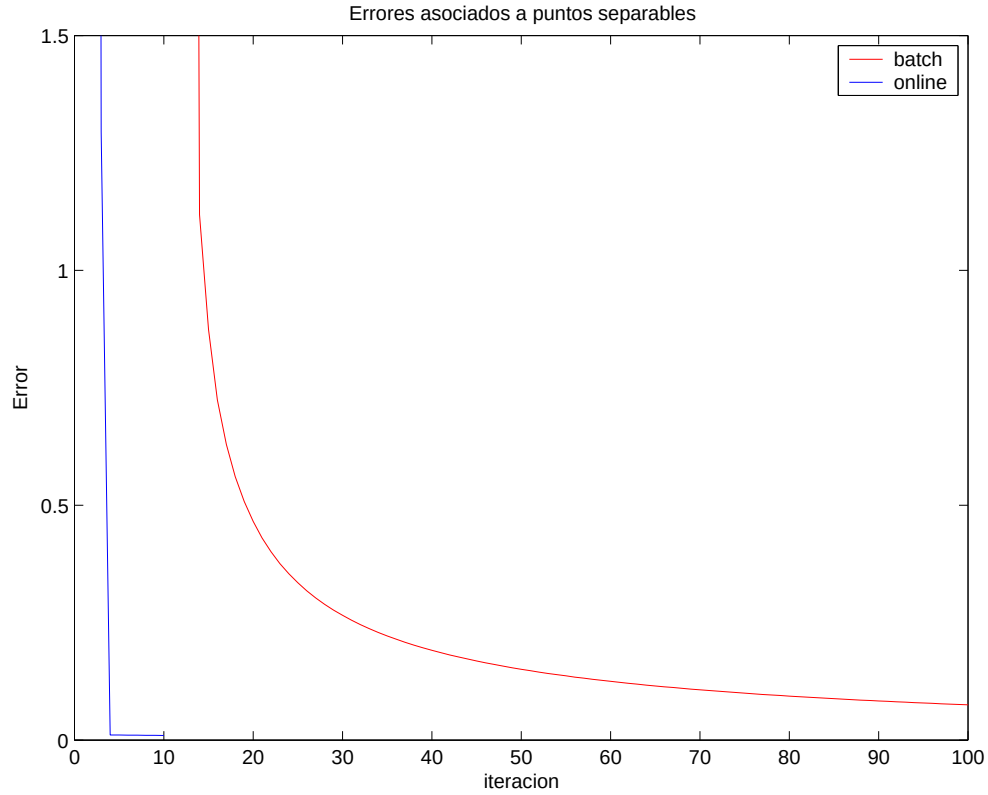
4.1. Gráficos

Para todos los experimentos tomamos los valores $mu = 0{,}01$ y $epsilon = 0{,}01$. (Se probó con *epsilon* menor, pero no cambiaba en nada el comportamiento general del problema). Acá se presentan los gráficos asociados al entrenamiento de los conjuntos mencionados en la sección anterior, y el último gráfico corresponde a un zoom del caso de los puntos linealmente separables, pues no se puede apreciar bien la diferencia entre amnos métodos a simple vista.


Errores asociados a puntos del "or" logico
batch
online
Error
iteracion
x 10^4
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3


Errores asociados a vocales vs consonantes
batch
online
Error
iteracion
x 10^4
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5


Errores asociados a puntos separables
batch
online
Error
iteracion
0
2
4
6
8
10
12
0
100
200
300
400
500
600
700
800



4.2. Resultados Numéricos

Los resultados numéricos de los pesos y la cantidad de iteraciones para cada caso es la siguiente:

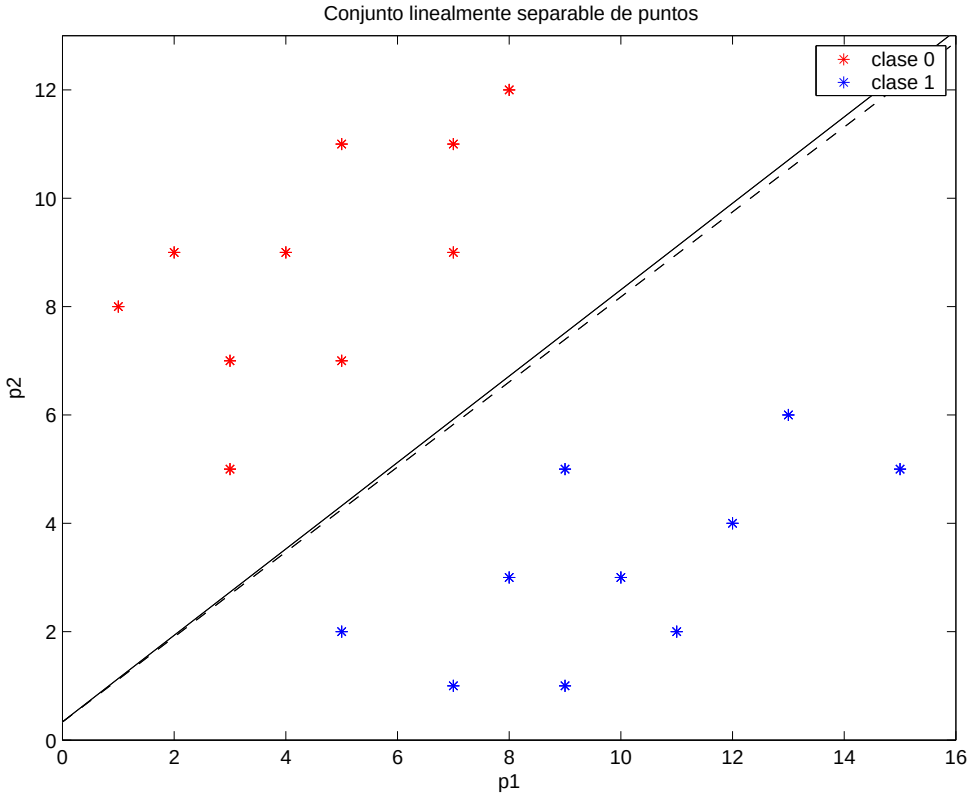
```
or
--
w_batch = -2.5107 5.5274 5.5273
it = 28559

w_online =-2.6546 5.6572 5.4301
it = 9150

letras
------
w_batch =    -0.0706 -1.6384 -3.5803 -2.3203 -3.4970 -0.6652
```

```
              -1.9829  1.5282  0.4345  1.4837 -3.0191 -0.1775
               0.1449  0.7831  0.6617 -1.3293 -0.2028  1.8769
               1.9619  1.9414  2.9182 -0.3123  1.6234  1.6015
               1.8434 -0.9568 -0.6377  0.2331  0.9991  0.2268
              -0.9752 -0.2381  3.6179  4.1982  4.3609  0.2339
it = 42463

w_online =   -0.0513 -1.6197 -3.6465 -2.3859 -3.5632 -0.7837
             -1.9642  1.5697  0.3936  1.5252 -3.1372 -0.1172
              0.1449  0.7422  0.6617 -1.4059 -0.1425  1.9807
              2.0248  2.0452  3.1819 -0.2521  1.6648  1.6020
              1.8849 -1.0334 -0.5775  0.2331  0.9582  0.2268
             -1.0518 -0.1779  3.5626  4.1019  4.3056  0.1569
it = 10987

puntos
------
w_batch = 0.4165 0.9827 -1.2325
it = 754

w_online = 0.4172 0.9812 -1.2510
it = 11
```

5. Conclusiones

En todos los cases se observa claramente que el error desciende mucho más rápido para el método de *entrenamiento on-line*. Impresionante es el último caso de los puntos separables en que el algoritmo de entrenamiento batch necesita 754 iteraciones, en tanto el algoritmo on-line con 11 llega al mismo resultado.

Los valores de los pesos obtenidos tambíen son prácticamente iguales. Para corroborar esto se muestra el gráfico de las rectas asociadas a los parámetros en el caso de los puntos linealmente separables, que es muy similar al caso del *or*:



Al considerar el cambio de gradiente respecto del último ejemplo visto en lugar de considerar el conjunto completo, el algoritmo on-line cláramente avanza más rápido que batch, **para los casos presentados**.

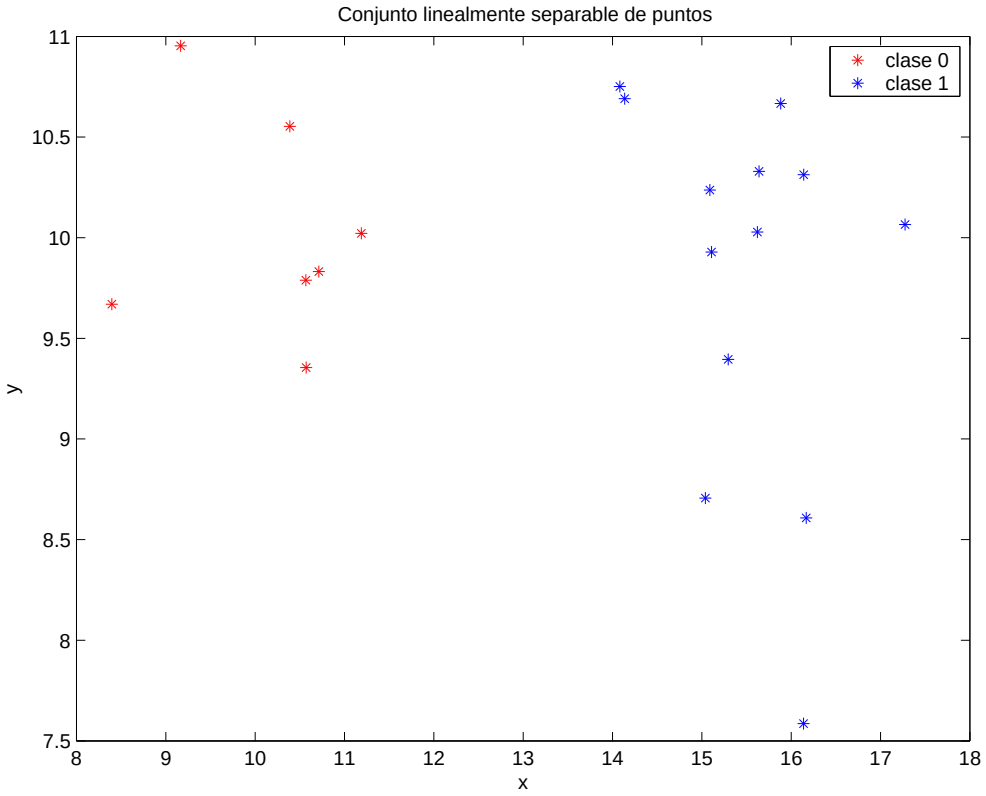
5.1. Caso especial

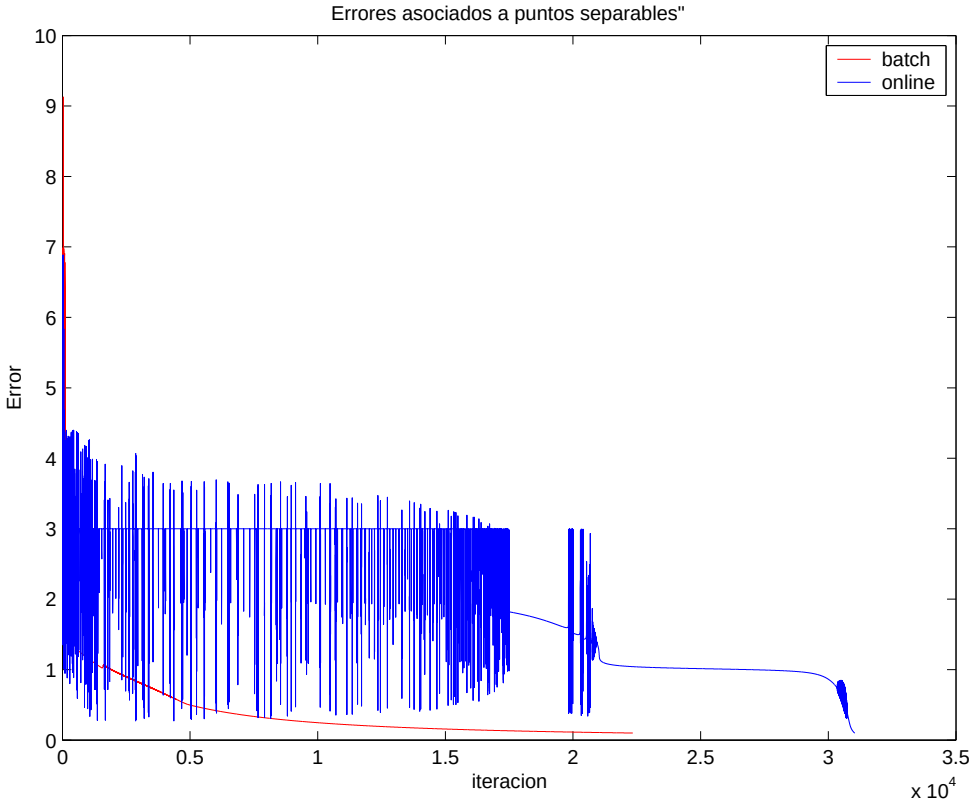
Sin embargo, con un conjunto encontrado en la web de puntos que a la vista son claramente linealmente separables, se obtuvo resultados bastante malos para el algoritmo on-line. Con este ejemplo podemos ver que el error total del algoritmo on-line oscila bastante en torno a mínimos locales, y que aunque finalmente converge, la dirección del gradiente indicada por cada ejemplo muchas veces no apunta al mínimo global. Se probó con distintas inicializaciones de w, valores de mu y $epsilon$, pero el resultado era similar o peor (no convergía para una gran cantidad de iteraciones).

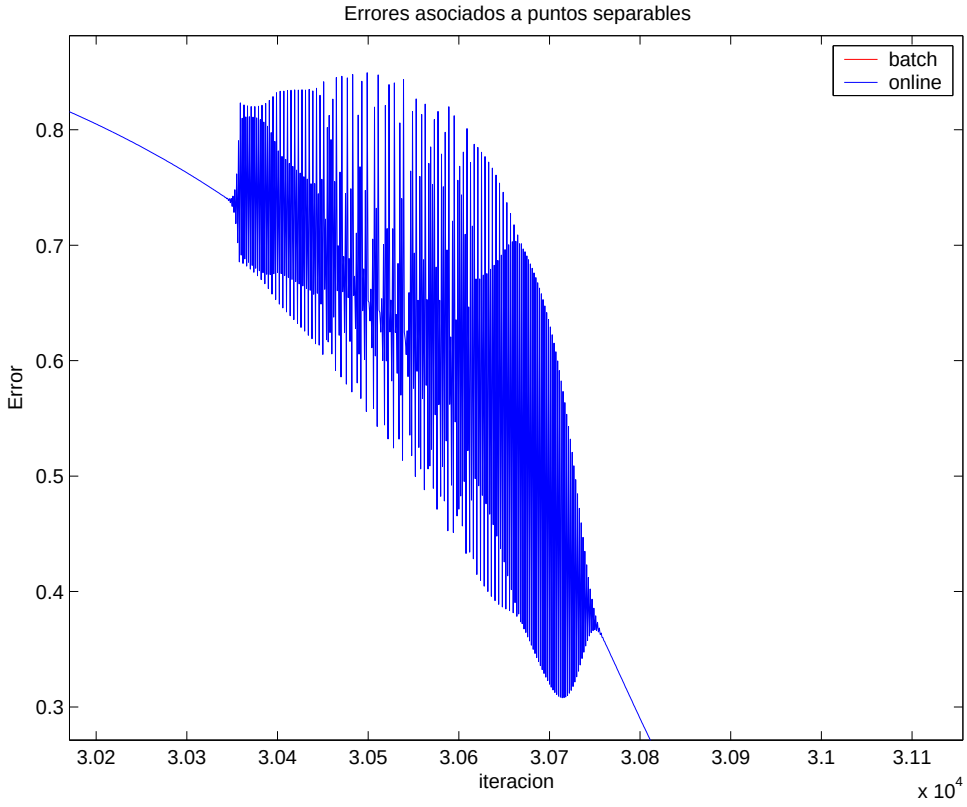
En el gráfico de las rectas que separan el espacio de los puntos, la recta del algoritmo on-line es la que pr'acticamente toca los puntos de ambos conjuntos (no así la recta del

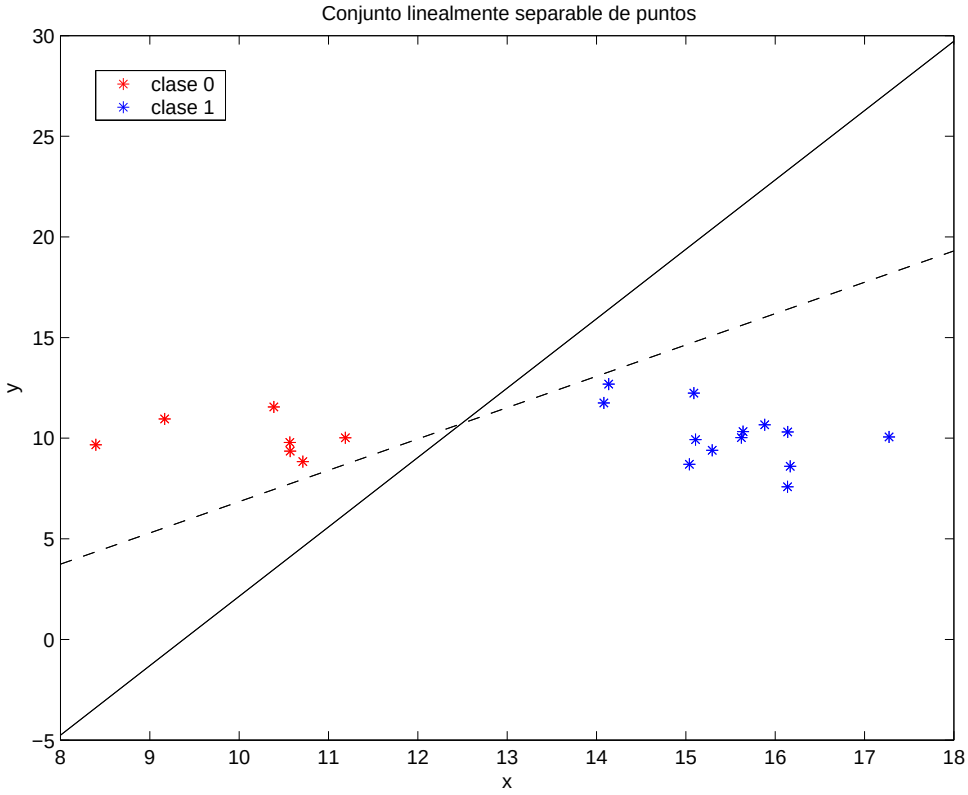
algoritmo batch). Esto nos indica que para nuevos elementos, la red neuronal entrenada por el algoritmo on-line probablemente se desempeñará peor que el algoritmo batch.

Tal vez este haya sido un ejemplo de laboratorio para demostrar que el algoritmo on-line no es siempre apropiado, o es un conjunto con características ocultas, pero la conclusión es que el algoritmo on-line para casos particulares puede ir demasiado rápido, aunque en general (para los simples casos que vimos) resulta ser mucho más rápido que el algoritmo batch.




Errores asociados a puntos separables"
batch
online
Error
iteracion
x 10^4
0
1
2
3
4
5
6
7
8
9
10
0
0.5
1
1.5
2
2.5
3
3.5


Errores asociados a puntos separables
batch
online
Error
0.8
0.7
0.6
0.5
0.4
0.3
3.02
3.03
3.04
3.05
3.06
3.07
3.08
3.09
3.1
3.11
iteracion
x 10^4




```
puntos
w_batch = -17.3019   1.8449 -0.5351
it =   41914
w_online = -25.8480   4.6168 -2.9660
it = 33592
```