DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico I

Aprendizaje supervisado

Redes Neuronales Artificiales

Primer Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Alvarez, Matías	090/12	matyy.alvarez@gmail.com
Levy Alfie, Jonás	081/12	jonaslevy5@gmail.com
Litwak, Brian	241/12	brian.litwak@gmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

Índice

1. Introducción y Teoría

Para esta primera entrega modelaremos casos de la vida real utilizando redes neuronales, pero más específicamente adentrándonos en el modelo de aprendizaje supervisado.

La idea será entrenar la red con información contenida en dos bases de datos, esperando que la misma aprenda rectificandose en cada paso de los errores que realiza respecto al resultado esperado, y experimentar variando valores internos como ser cuántas neuronas utilizar, con qué rapidez se quiere que la red aprenda, durante cuántas épocas aprender, entre otros. Finalmente el trabajo tratará de extraer conclusiones sobre lo experimentado.

A continuación presentamos una breve introducción con las nociones básicas y generales del modelo utilizado.

1.1. Perceptrón multicapa

La idea principal del perceptrón multicapa es utilizar el paradigma de aprendizaje supervisado con un algoritmo de corrección de error. El aprendizaje supervisado se basa en un entrenamiento en el cual se provee al sistema con información de las entradas y de igual forma se proveen las salidas esperadas para cada entrada en particular. Luego, dado que se quiere que el perceptrón aprenda de la mejor forma posible, se evalúa en cada paso su desempeño y se contrasta con el resultado que tuvo que haber arrojado, para que así el sistema se de cuenta de cuánto tiene que modificar sus valores para que esta diferencia se vuelva menor. Intuitivamente, el perceptrón multicapa permite aproximar funciones, categorizar y encontrar patrones.

El concepto de neurona artificial, en el cual se basa el perceptrón multicapa, parte de los principios neuro-biológicos que describen el comportamiento de las neuronas en la corteza cerebral. Una neurona artificial tiene elementos de entrada y salida que se procesan en una unidad central y otros elementos que permiten que la neurona generalice y aprenda conceptos.

La estructura de una neurona se puede representar de la siguiente manera.

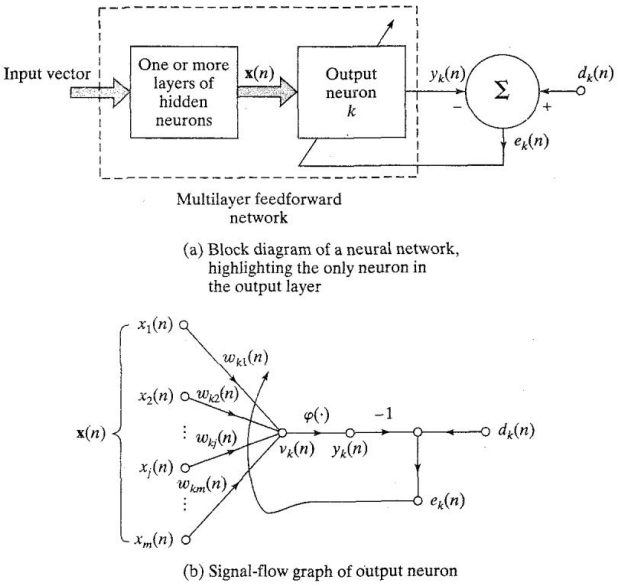

Figura 1: Esquema de neurona artificial

En dicha imagen, tomada de la bibliografía[1], podemos observar como los elementos más importantes al vector de entrada de la neurona $(x_1, ..., x_m)$, a los pesos correspondientes como w_{ij}, a la función de activación $\phi()$ y al elemento de salida. A partir de esta estructura básica la neurona puede mapear las entradas para obtener a la salida una respuesta deseada que pudiera pertenecer a alguna función a determinar.

La función de activación trata de simular el mecanismo que realiza el sistema de neuronas en el cerebro, que se basa (como un caso más sencillo) en la exitación de las neuronas hasta un cierto punto en el cual se pasa un umbral en el que dicha neurona dispara la información que le corresponde. De la misma manera se pueden utilizar otras funciones de activación más sofisticadas y cada una de ellas generará un resultado distinto en la distribuición de los pesos, por ende en el resultado final.

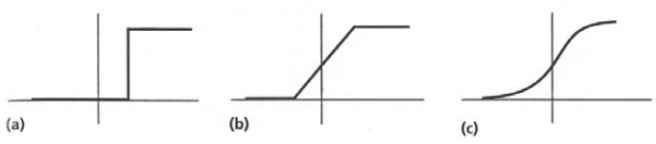

Figura 2: Algunas funciones de activación

La red neuronal del perceptrón cuenta con tres funciones básicas:

- Una función de *activación* que mapea los pesos de las respectivas conexiones que tiene una neurona en la salida de la misma.
- Una función de *corrección* que calcula las diferencias de los pesos de las conexiones entre las neuronas, comparando el resultado que se obtuvo al hacer la activación con el resultado esperado (se puede notar a estas diferencias como $\triangle W$).
- Una función de *adaptación* que realiza el cambio de los pesos anteriores con los nuevos pesos y depende del modo con el que se esté operando: si se trata de aprendizaje incremental, se realiza por cada caso de entrenamiento (luego de cada corrección); de lo contrario si es batch se hace todo junto al finalizar todas las correcciones.

Por otro lado el perceptrón cuenta con un *coeficiente de entrenamiento* (notado como η) que indica qué tanto varían los pesos entre iteración e iteración, por lo cual indica qué tan lento o rápido la red se entrena.

Finalmente la diferencia clave que hay entre este perceptrón y el simple es que éste cuenta con varias capas extra de neuronas que se pueden colocar entre las neuronas de entrada y de salida. El objetivo de esto es poder aprender con un mayor nivel de complejidad y resolver problemas que de la otra forma no tendrían solución (por ejemplo, con problemas no lineales como ser la función *XOR*). Por lo cual en nuestro caso se pueden emplear n capas ocultas internas, cada una con m_i neuronas ($i \in \{1, ..., n\}$).

[1] Simon Haykin - Neural Networks A Comprehensive Foundation - Página 74

Dicho esto, el esquema del perceptrón multicapa se basa en iniciar la red neuronal con pesos (que suelen ser aleatorios uniformemente distribuidos), entrenar con el dataset que se utilice realizando en orden la activación, corrección y adaptación las veces que uno considere necesario hasta que el problema otorgue una buena solución.

Entre algunos problemas que puede presentar está el *overfitting*, lo cual se basa en sobre-entrenar a la red con los mismos casos de entrenamiento y que sólo consiga responder correctamente a los mismos, por lo que si se quiere obtener el resultado para entradas que la red nunca vió, éste podría ser muy malo. Esto básicamente es afectar la capacidad de generalización de la red. Este problema puede darse por dos motivos: un aprendizaje demasiado largo o bien una red con muchos nodos y capas.

Entre algunos usos prácticos que se le puede dar al perceptrón multicapa están la compresión de datos, la resolución de problemas de asociación de patrones, entre otros.

2. El problema

En este trabajo práctico se utilizará el modelo de perceptrón multicapa (usando el método de retro-propagación del error). Para esto se asignan dos bases de datos referentes a problemas reales y el objetivo es analizar qué configuraciones de estas redes neuronales resultan mejores para cada una, desde el punto de vista de la generalización de datos.

2.1. Los datos del problema

El primer conjunto de datos se trata sobre diagnósticos de *cancer de mamas* en donde, además de los datos y características de los pacientes, se sabe fehacientemente si cada uno tiene o no dicha enfermedad. Cada entrada corresponde a los datos de distintos pacientes y contiene 30 características provenientes de imágenes digitalizadas de muestras de células. Además de esto, se tiene el atributo del diagnóstico final que indica si la muestra pertenecía a un tumor maligno o benigno. Dado que el único atributo que no es un número es el diagnóstico, pasamos a representar los valores benignos como un 0 y los malignos como un 1. Sobre este primer dataset se buscará entrenar una red neuronal para poder predecir, dado un nuevo paciente con sus datos, si su estado será el de tener cancer o no, es decir, estamos frente a un problema de clasificación.

Por otro lado, el segundo conjunto de datos se trata sobre requerimientos de *carga energética* para calefaccionar y refrigerar edificios en función de ciertas características de los mismos. El conjunto de datos se generó usando edificios de distintas formas que difieren respecto a la superficie y distribución de las áreas de reflejo, orientación y otros parámetros (en total hay 8 parámetros que consideramos de entrada). Además, se sabe fehacientemente para cada uno de estos edificios la cantidad de energía que fue necesaria para realizar una calefacción y refrigeración adecuadas. Sobre este segundo dataset se buscará entrenar una red neuronal para poder aproximar la función que determina la cantidad de energía necesaria para la calefacción y refrigeración de un edificio en función de los parámetros de entrada. Es decir, es un problema de regresión.

2.2. Implementación

La implementación se realizó en python ya que es el lenguaje de programación sugerido por la cátedra sobre el cual tenemos mayor conocimiento. Utilizamos *numpy* y *matplotlib* como librerías para operaciones sobre matrices o graficar.

Realizamos la versión sin momentum del algoritmo de backpropagation ya que nos basamos fuertemente en el pseudocódigo propuesto por la cátedra. Por ende, nuestra red va a converger más lentamente a diferencia de la versión con momentum.

Por lo dicho en la parte teórica, la matriz de pesos se inicializa de forma aleatoria con pesos uniformemente distribuidos y se tienen tantas matrices de pesos (L) como cantidad de capas internas que tiene la red más 1. Ademas, inicializamos cada red neuronal con un valor de coeficiente de aprendizaje y con una función de activación.

Utilizaremos dos funciones de activación:

- Binomial
- Bipolar

Como mencionamos en la parte teórica, la red neuronal consta de tres funciones básicas (Activación, Corrección, y Adaptación) las cuales utilizamos la sugerencia de la cátedra para su implementación. A continuación el pseudocódigo de las funciones mencionadas anteriormente pero con una pequeña variación en correction para que sea más facil el paso a la implementación.

Algorithm 1 Pseudo-código para Activation(X_h)

$Y_0 = X_h$
for $j \in \{1, ..., L\}$ **do**
    $Y_j = funcionActivacion((Y[j-1]o1) \times W_j)$
end for
ret $Y[L]$

Aquí '$\times$' representa la multiplicación de matrices y o es la concatenación de un elemento al final. Además, $.T$ es la transpuesta de una matriz.

Algorithm 2 Pseudo-código para Correction(Z_h)

$Error = Z_h - Y[L]$
$e = \sum_i (Error[i] * Error[i])$
(Primero calculamos todos los E y luego los $\triangle W$)
(Calculamos los E)
$E = [Error * funcionActivacionPrima(Y[L])]$
for $j \in \{L-1, ..., 1\}$ **do**
    $E.agregar((E[-1] \times W[j].T) * funcionActivacionPrima(Y[j]))$
end for
$E.darVuelta$
(De ahora en más es más facil trabajar con los índices)
(Calculamos los $\triangle W$)
for $j \in \{0, ..., L-1\}$ **do**
    $\triangle W[j] = \triangle W[j] + coeficienteAprendizaje * (Y[j].T \times E[j])$
end for
ret e

Por último la función de adaptación queda definida como sigue.

Algorithm 3 Pseudo-código para Adaptation()

```
for j ∈ {1, ..., L} do
    W[j] = W[j] + △W[j]
    △W[j] = 0
end for
```

Existen dos formas de entrenar a una red neuronal dado un conjunto de test.

- Incremental: A cada test, se le realiza activation, correction y adaptation en ese orden para luego continuar por el siguiente test.
- Batch: A cada test se le realiza activation y correction. Cuando se haya realizado lo anterior para todos, se hace un único adaptation para todos los test.

En ambas formas de entrenar, la bibliografía sugiere que se cambie el orden de recorrer los test en cada época por ser estocásticos.

Presentaremos a continuación las dos formas de entrenar a una red neuronal en forma de pseudocódigo.

Algorithm 4 Pseudo-código para Incremental(X, Z)

```
pos = [i for i in range(p)]
Permutar(pos)
e = 0
for h ∈ {1, ..., P} do
    activation(X[pos[h]])
    e = e + correction(Z[pos[h]])
    adaptation()
end for
ret e
```

Algorithm 5 Pseudo-código para Batch(X, Z)

```
pos = [i for i in range(p)]
Permutar(pos)
e = 0
for h ∈ {1, ..., P} do
    activation(X[pos[h]])
    e = e + correction(Z[pos[h]])
end for
adaptation()
ret e
```

Por último, implementamos los pseudo-códigos que nos dió la cátedra de *validación_none* y *holdout* como posibles formas de validaciones.

Esto último realiza *entrenar*, que puede ser el llamado a la función *incremental* o bien a *batch*.

En donde *testing* es una función que se representa con el siguiente pseudo-código.

El código se puede encontrar en el archivo con nombre *helpme.py* donde también se encuentra el almacenamiento de datos y generación de gráficos.

Algorithm 6 Pseudo-código para validacion_none($X, Z, errorMin, maxEpocas$)

1: $error = 1$
2: $epoca = 0$
3: **while** $error > errorMin \wedge epoca < maxEpocas$ **do**
4: $\quad error = entrenar(X, Z)$
5: $\quad epoca = epoca + 1$
6: **end while**
7: ret $error, epoca$

Algorithm 7 Pseudo-código para Holdout($X, Z, errorMin, maxEpocas, r$)

1: $T, V = particionar(X, Z, r)$
2: $errorTesting = 1$
3: $errorEntrenamiento = 1$
4: $epoca = 0$
5: **while** $errorEntrenamiento > errorMin \wedge epoca < maxEpocas$ **do**
6: $\quad errorEntrenamiento = entrenar(T_X, T_Z)$
7: $\quad errorTesting = testing(V_X, V_Z)$
8: $\quad epoca = epoca + 1$
9: **end while**
10: ret $errorEntrenamiento, errorTesting, epoca$

Algorithm 8 Pseudo-código para Testing(X, Z)

1: $error = 0$
2: **for** $h \in \{1, ..., len(X)\}$ **do**
3: $\quad y = activation(X[h])$
4: $\quad error = error + ((Z[h] - y) * (Z[h] - y))$
5: **end for**
6: ret $error$

2.3. Opciones del programa y modo de uso

La implementación acepta tres parámetros, se ejecuta por línea de comando con el programa `python`. El modo de uso es el siguiente:

`python tp2ej1.py [bd] [coeficienteAprendizaje] [errorMínimo] [épocasEntrenamiento] [modo] [tamMuestra] [funciónSigmoidea] [cantidadCorridas] [cantidadCapasInternas] [nodosCapa1] ... [nodosCapaN]`

- `bd` indica qué base de datos usar. En caso de querer usar la base de datos de diagnóstico de *cáncer de mamas*, se deberá ingresar un 1. En caso de querer usar datos de *eficiencia energética*, se deberá ingresar un 2.
- `coeficienteAprendizaje` debe ser un valor entre 0 o 1 e indica el valor del cambio en los $\triangle W$.
- `errorMínimo` es un número positivo, que representa el error mínimo que se desea obtener al entrenar la red neuronal (es un criterio de parada, por lo que si se quiere ejecutar una cantidad fija de épocas, deberá ser un valor muy pequeño).
- `épocasEntrenamiento` es la cantidad máxima de épocas que se desea utilizar para entrenar la red neuronal.
- `modo` usa valores $\{1, 2, 3, 4\}$ donde se indica la la forma de entrenar a la red neuronal (sea incremental o batch) y si usa validación o no. El valor 1 indica holdout incremental, el valor 2 indica holdout batch, el valor 3 indica sin validación con incremental y el valor 4 indica sin validación con batch.
- `tamMuestra` debe ser un número real entre 0 y 1. Es el porcentaje de muestra que se utilizará para realizar el training.
- `funciónSigmoidea` debe tomar el valor 1 cuando se quiere utilizar la función binomial mientras que debe tomar el valor 2 para la bipolar.
- `cantidadCorridas` es un número positivo entero que indica la cantidad de veces que se desea ejecutar el programa.
- `cantidadCapasInternas` es un número natural que indica la cantidad de capas internas que tiene la red neuronal seguido por la cantidad de nodos que tiene cada capa. Es decir, si tenemos 3 capas internas con 5 nodos en la primera, 3 en la intermedia y 2 en la última capa, luego del parámetro de `cantidadCapasInternas` seguirán los parámetros para `nodosCapa1`, `nodosCapa2` y `nodosCapa3` (siendo el de este ejemplo 3 5 3 2).

El programa muestra por la consola por cual época se encuentra con el error obtenido y tiempo que demandó esa época.

Además el programa genera varios archivos de salida:

- Un archivo de texto (`errorEntrenamiento[i].txt`): se crea un archivo de esto para cada corrida, donde el [i] indica el número de corrida que le corresponde. En este archivo se guarda para cada época la suma de los errores obtenidos del conjunto de entrenamiento.
- Un archivo de texto (`errorNEntrenamiento[i].txt`): al igual que el anterior, se crea un archivo de esto para cada corrida. En el archivo se guarda para cada época, el promedio de los errores obtenidos del conjunto de entrenamiento.
- Un archivo de texto (`errorTesting[i].txt`): se crea un archivo de esto para cada corrida, donde el [i] indica el número de corrida que le corresponde. En este archivo se guarda para cada época la suma de los errores obtenidos del conjunto de testing.

- Un archivo de texto (`errorNTesting[i].txt`): al igual que el anterior, se crea un archivo de esto para cada corrida. En el archivo se guarda para cada época, el promedio de los errores obtenidos del conjunto de testing.
- Un archivo de texto (`tiempos[i].txt`): al igual que el anterior, se crea un archivo de esto para cada corrida donde para cada época, se almacena el tiempo consumido por la misma.

Usamos todos estos recursos para la experimentación de este trabajo.

3. Experimentación sobre el dataset de *Cancer de mamas*

Para la experimentación sobre este dataset se fueron planteando una serie de pruebas con el objetivo de encontrar configuraciones de los parámetros que den buenos resultados. A medida que obteníamos dichos resultados fuimos planteando nuevos experimentos para seguir mejorando los resultados.

Principalmente la experimentación se dividió en dos, usando por un lado entrenamiento *batch*, y por el otro *incremental*.

Probamos para cada modo varias configuraciones, variando el *learning rate* (η), el tipo de función sigmoidea. Aclaramos aquí que, dado que los resultados usando la función bipolar fueron siempre peores en comparación a la binaria, no incluimos resultados usando la sigmoidea bipolar.

Otro aspecto muy importante de la configuración de una red neuronal es su arquitectura o topología, es decir la *forma* que tendrá. Está claro que la red tendrá tantas neuronas de entrada como campos de entrada haya en el dataset, y tendremos una única neurona de salida cuyo resultado usaremos para clasificar.

Lo que queda definir son las capas intermedias de neuronas, y la cantidad de neuronas de cada una. En base a lo que sabemos, la mayoría de los problemas pueden resolverse decentemente usando una única capa intermedia, y dada la simplicidad del modelo y el tiempo acotado para probar demasiadas combinaciones, nos limitaremos al mismo. Finalmente, nuestra red para estos experimentos tendrá siempre una capa intermedia, y otro de los parámetros a variar será la cantidad de neuronas en dicha capa.

En la experimentación, probamos usando 5, 10, 15, 20 y 25 neuronas para todos los experimentos. Se usó un 80 % del dataset para el entrenamiento y el 20 % para la validación. Las muestras para esto las selecciona el programa al azar.

A continuación detallamos cada experimento, mostramos y analizamos sus resultados.

3.1. Resultados y Análisis

Primera prueba

Lo primero que hicimos fue probar ambos modos de entrenamiento batch e incremental, con un valor de η que parecía razonable para empezar. Corrimos cada modo, usando las cantidades de neuronas mencionadas. A cada configuración se la dejó correr por 5000 épocas, para poder darnos una idea de cómo se comporta en un lapso considerable de entrenamiento.

Medimos para cada época el error y el tiempo consumido para esa época. Estos fueron los resultados.

Mostramos primero que nada gráficos de la suma de errores para batch e incremental en función de la época. Lo que llamamos suma de errores, es la suma de los cuadrados de las diferencias entre el valor obtenido y el valor deseado.

Entrenamiento batch:

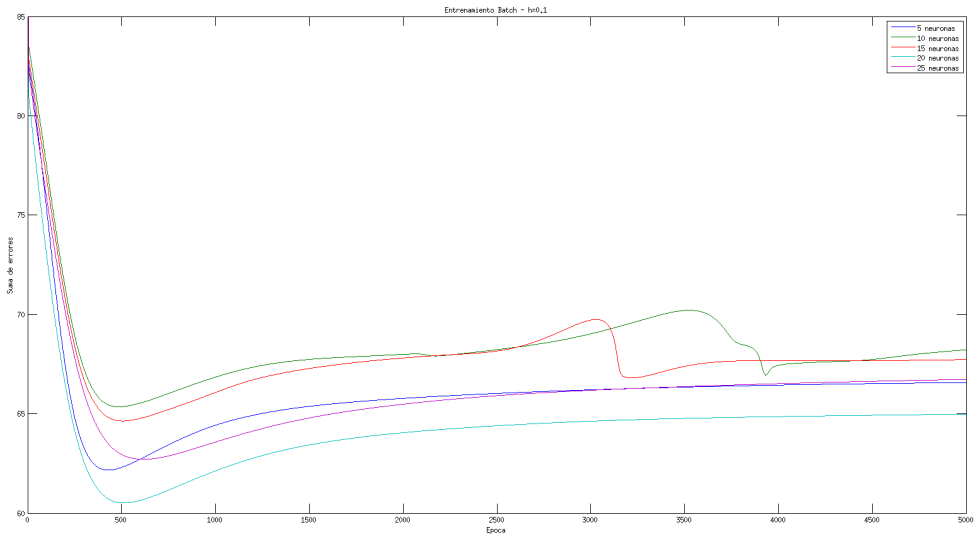

Figura 3: Suma de errores vs. Épocas - Entrenamiento Batch - $\eta = 0{,}1$

Validación batch:

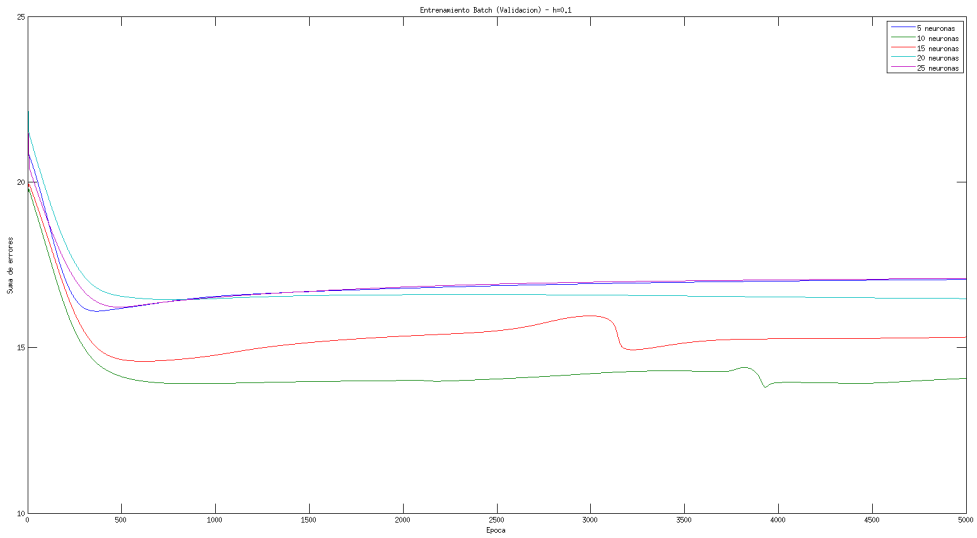

Figura 4: Suma de errores Validación vs. Épocas - Entrenamiento Batch - $\eta = 0{,}1$

Entrenamiento Incremental:

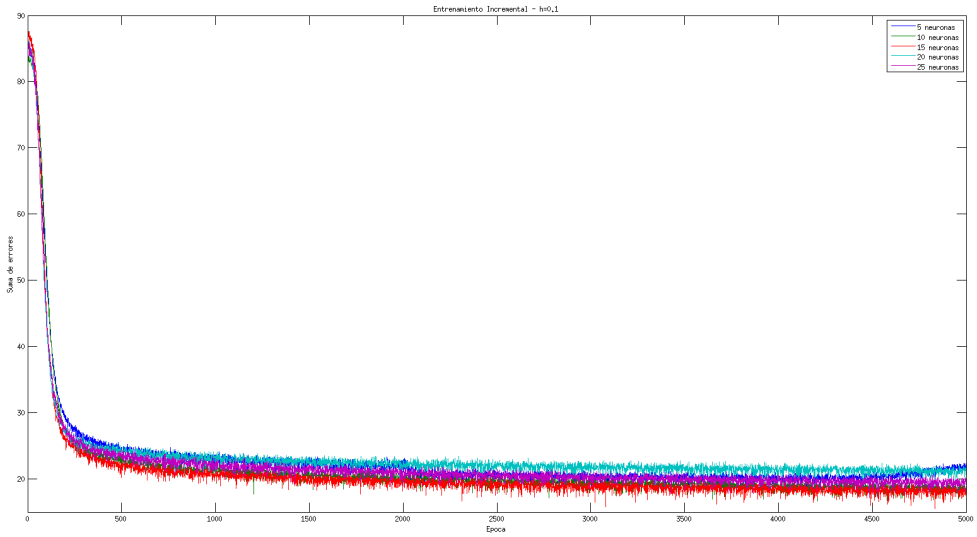

Figura 5: Suma de errores vs. Épocas - Entrenamiento Incremental - $\eta = 0{,}1$

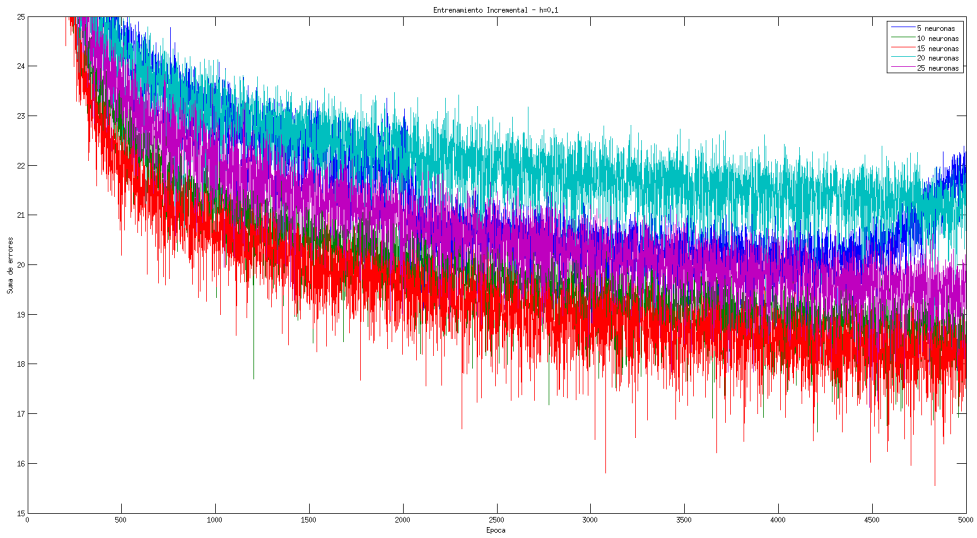

Figura 6: Suma de errores vs. Épocas - Entrenamiento Incremental - $\eta = 0{,}1$ (Zoom)

Validación incremental:

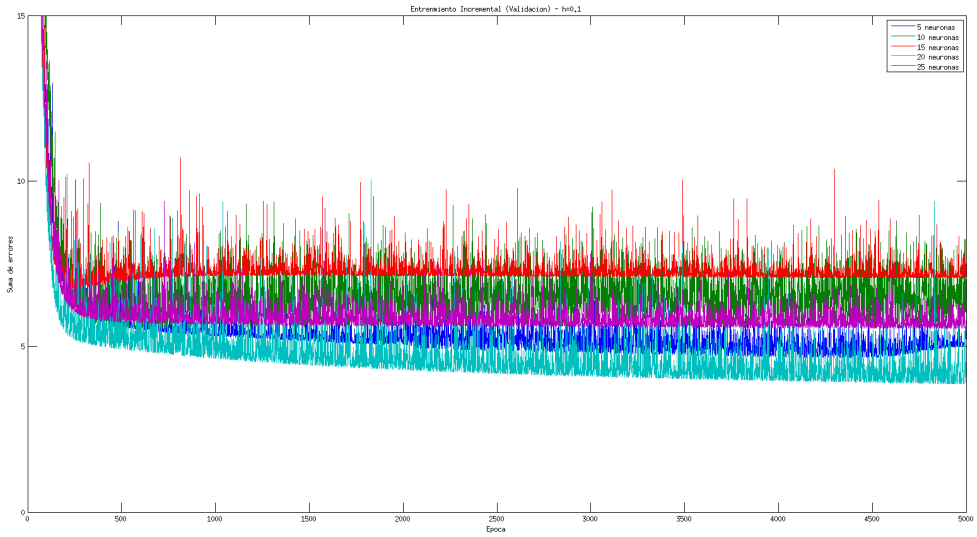

Figura 7: Suma de errores (Validación) vs. Épocas - Entrenamiento Incremental - $\eta = 0{,}1$

Dada la notoria diferencia entre los tiempos de ambos modos, nos parece pertinente incluir un gráfico para compararlos.

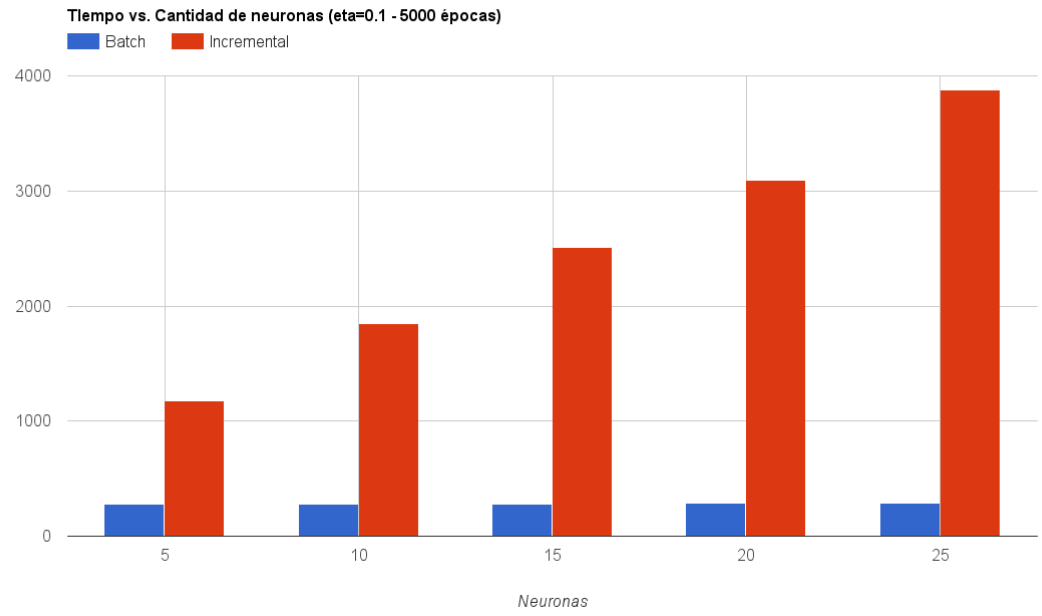

Figura 8: Tiempos Batch vs. Incremental

Podemos notar hasta aquí varias cosas. Primero que nada, los resultados finales de errores usando incremental fueron mucho mejores que usando batch. Más aún, no parece que $\eta = 0{,}1$ sea un muy buena elección para este último modo, dado que viendo su gráfico podemos ver que primero logra alcanzar un mínimo cerca de la época 500 de aproximadamente 60 (con 20 neuronas), pero luego se aleja de este valor y queda en 65. Con las otras cantidades de neuronas sucede de igual manera, salvo que además algunas muestran oscilaciones.

Para las corridas con modo incremental, en todas hay oscilaciones muy pequeñas y permanentes a lo largo de todas las series. Esto probablemte sea algo inherente al método incremental, dado que en cada época se realizan tantas adaptaciones como épocas, entonces el crecimiento del error se vuelve más inestable e impredecible, a diferencia de batch. Sin embargo en líneas generales, las series muestran un decrecimiento constante salvo en el caso que se usan 5 neuronas, y se alcanza un resultado de error final de aproximadamente 18, y de todas formas las oscilaciones son pequeñas.

Los gráficos con los errores de validación se mantienen muy similares a sus correspondientes, excepto por el rango de valores de los mismos, de modo que no incluiremos en el futuro de la experimentación gráficos con errores de validación salvo que se encuentren grandes diferencias (todos los gráficos no presentes en este informe puede verse en la carpeta de la entrega).

Hasta aquí hemos obtenido resultados bastante mejores usando el modo incremental, pero viendo la figura 8 queda claro que fue a un costo bastante alto. De este gráfico vemos como crecen en gran medida los tiempos totales del entrenamiento usando el modo incremental, a medida que aumenta la cantidad de neurona en la capa intermedia. Incluso usando una cantidad chica de neuronas (5), el modo batch tarda muchísimo menos.

Vemos que el modo incremental es mucho más sensible a la cantidad de neuronas que haya en la red, y esto tiene bastante sentido, dado que, como ya mencionamos antes, se realizan adaptaciones por muestra en cada época, haciendo que cada en cada época se realicen muchas adaptaciones, que implican modificar los pesos de una matriz, la cual además aumenta en dimensión si aumenta la cantidad de neuronas. En cambio en batch, se realiza una sola adaptación por época, y podemos observar que esto le permite que el tiempo total variando la cantidad de neuronas se vea casi inmutado.

Con todo esto, planteamos dos nuevas pruebas. Una para poder encontrar configuraciones que nos

den mejores resultados usando el modo batch (en lo posible tan buenos como los obtenidos con incremental). En la otra prueba, veremos qué sucede con otros valores de η para el modo incremental, para ver si todavía podemos mejorar los resultados en términos de error.

Segunda prueba

Para esta segunda prueba trataremos de mejorar lo más posible los resultados en términos de error para el modo batch. En base a lo observado, nos pareción que lo que podría estar sucediendo es que el valor de η sea demasiado grande, y esté causando que la solución no converja bien.

Decidimos entonces probar usando valores más chicos, y probamos con $\eta = \{0{,}1, 0{,}05, 0{,}01, 0{,}005, 0{,}001\}$. Las cantidades de neuronas se mantuvieron como antes. Mostramos ahora los resultados del error en función de la época, separados en gráficos por cantidad de neuronas, es decir un gráfico contendrá las mediciones para una cantidad de neuronas específica con los 5 valores de η planteados.

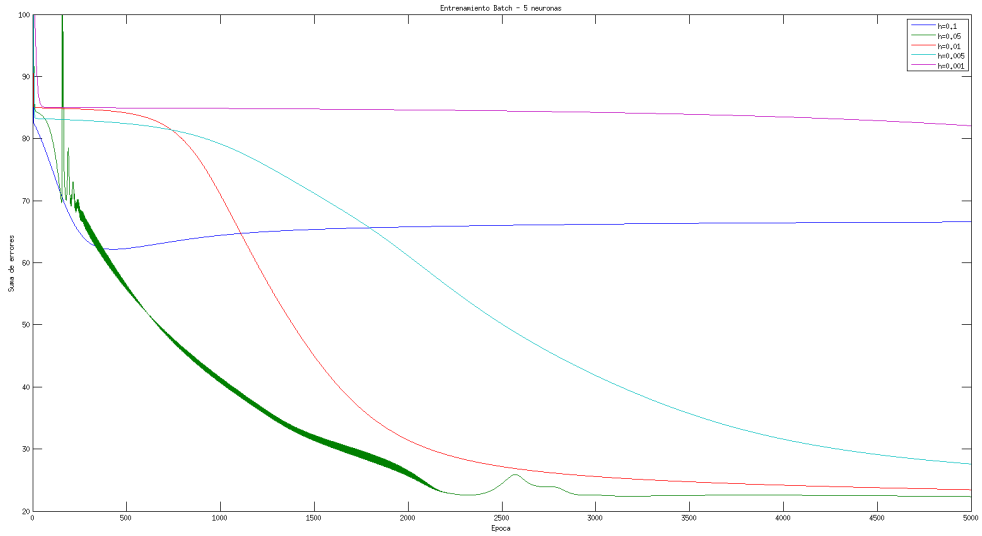

Figura 9: Suma de errores vs. Épocas - Entrenamiento Batch - 5 neuronas

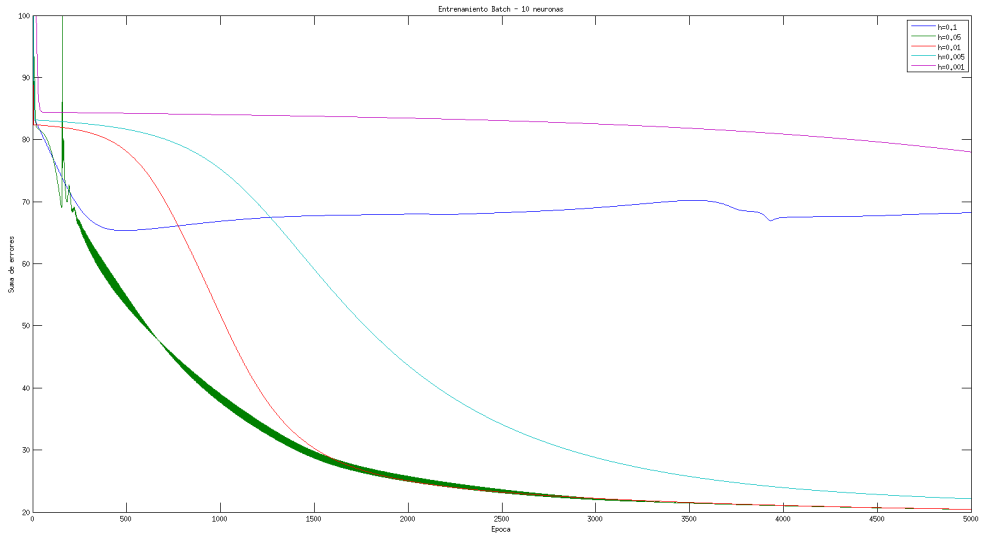

Figura 10: Suma de errores vs. Épocas - Entrenamiento Batch - 10 neuronas

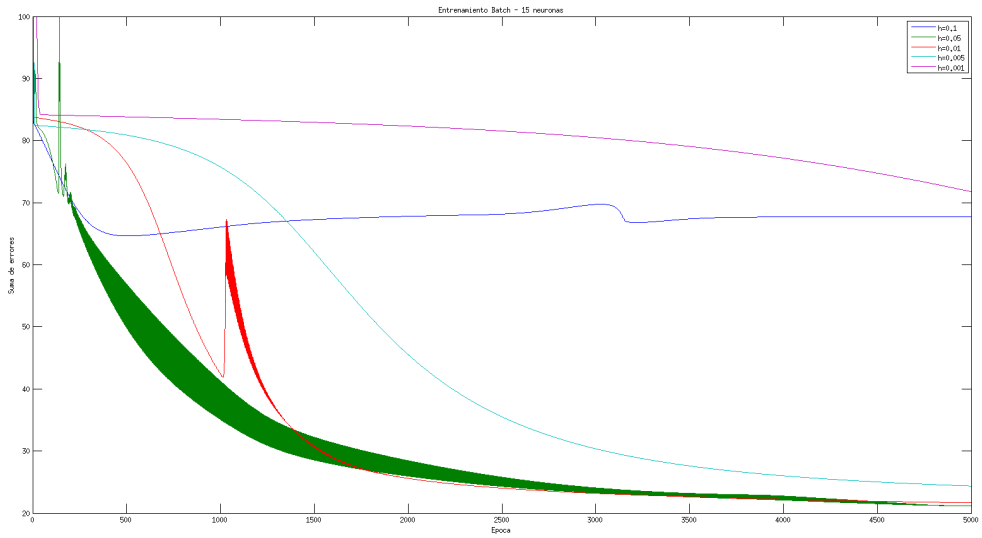

Figura 11: Suma de errores vs. Épocas - Entrenamiento Batch - 15 neuronas

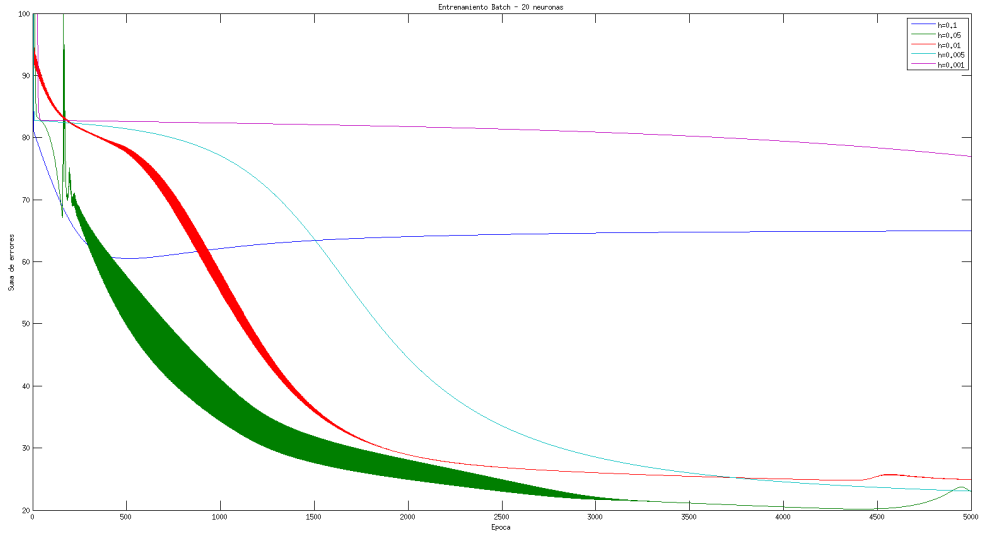

Figura 12: Suma de errores vs. Épocas - Entrenamiento Batch - 20 neuronas

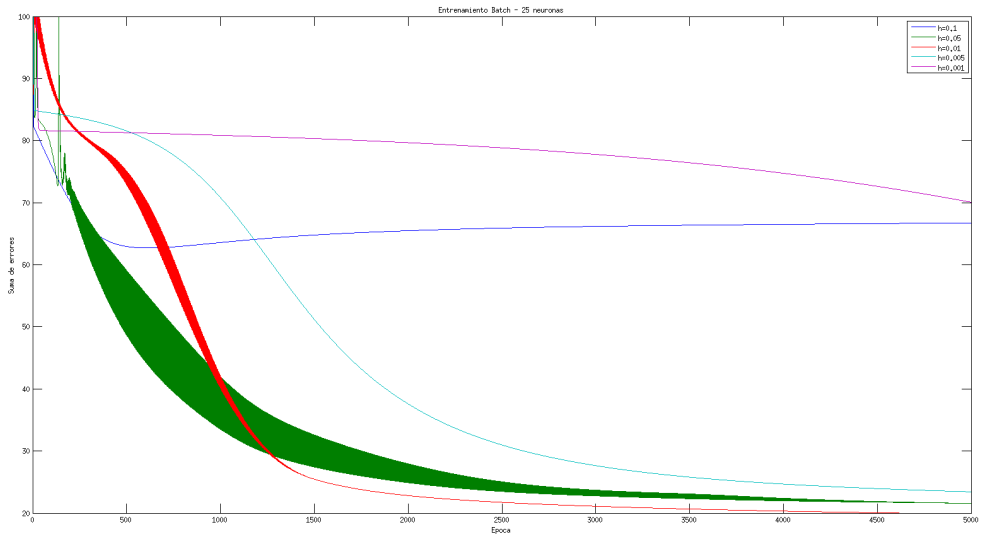

Figura 13: Suma de errores vs. Épocas - Entrenamiento Batch - 25 neuronas

Viendo estos resultados, podemos ver primero que todos los gráficos guardan cierta similitud entre ellos. No incluímos gráficos de tiempo, dado que como vimos antes, varía prácticamente el tiempo usado para el entrenamiendo con modo batch cambiando la cantidad de neuronas, y esto se mantuvo así en esta experimentación también (variar el valor de η tampoco presenta cambios).

Con este experimento, logramos conseguir para algunos valores de η resultados bastante buenos. Incluimos los resultados anteriores con $\eta = 0{,}1$ para poder compararlos. En particular, usando η $0{,}05$ o $0{,}01$ se obtienen muy buenos resultados. Ya con $0{,}005$ el error que se consigue no es tan bajo, y con $0{,}001$ sube todavía más y decrece muy lentamente, probablemente porque este valor ya es demasiado chico.

Todas las corridas usando, η $0{,}05$ o $0{,}01$ dan resultados de error finales de entre 25 aproximadamente hasta 18 (usando 25 neuronas). Con estos valores vemos que podemos lograr obtener valores de errores bastante bajos y muy cercanos a los obtenido en la experimentación anterior con modo incremental, y cabe recordar que con un costo temporal muchísimo más bajo.

Respecto a la forma de algunos gráficos, vemos que justamente usando estos valores las mediciones oscilan bastante, formando estas especies de curvas con volumen. Parecería además que con más neuronas en la capa intermedia esta oscilación se hace más grande en las épocas. Sin embargo, la forma en que oscilan algunas de estas series es más *suave* de alguna forma, distinto a como oscilaban las corridas de modo incremental.

Finalmente, de este experimento, podemos concluir que el valor de η elegido en un primer lugar no era el más adecuado, y usando valores entre 0.05 y 0.01 se obtienen muy buenos resultados con el mismo tiempo que las pruebas anteriores (5000 épocas en menos de 5 minutos de entrenamiento), y estos resultados tienen una calidad como las corridas con modo incremental.

Tercera prueba

Como dijimos antes, trataremos de ver si podemos mejorar aún más el desempeño del modo incremental, y dado el elevado tiempo de cómputo que requiere, trateremos de encontrar un configuración que sea muy superior en términos de error a lo logrado anteriormente.

Mantuvimos las cantidades de neuronas anteriores, y ahora probamos con los valores de $\eta = \{0{,}1, 0{,}2, 0{,}3\}$.

Mostramos ahora los resultados.

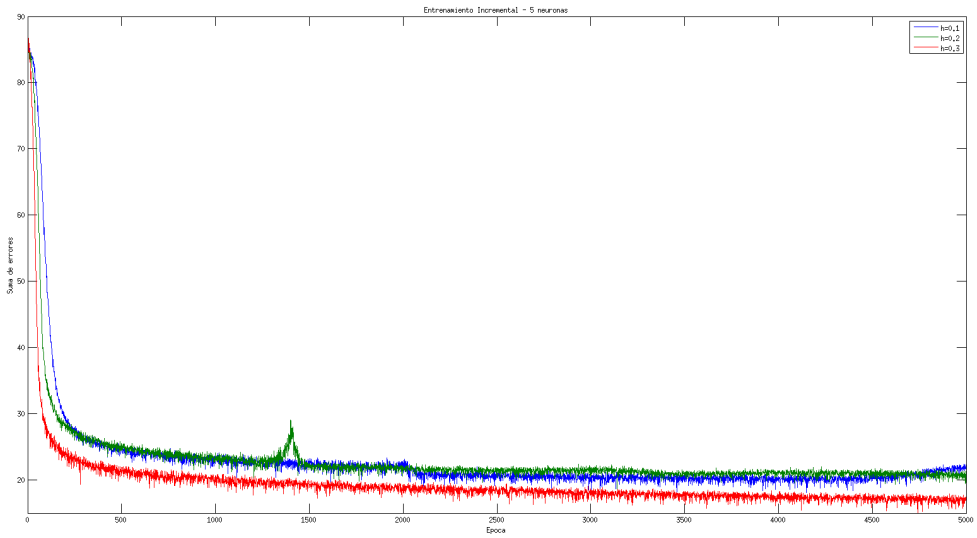

Figura 14: Suma de errores vs. Épocas - Entrenamiento incremental - 5 neuronas

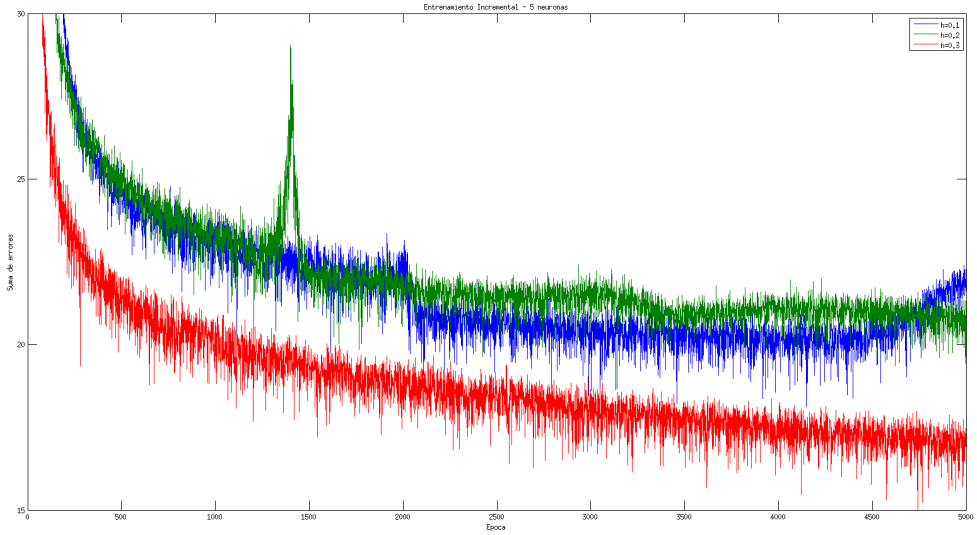

Figura 15: Suma de errores vs. Épocas - Entrenamiento incremental - 5 neuronas (Zoom)

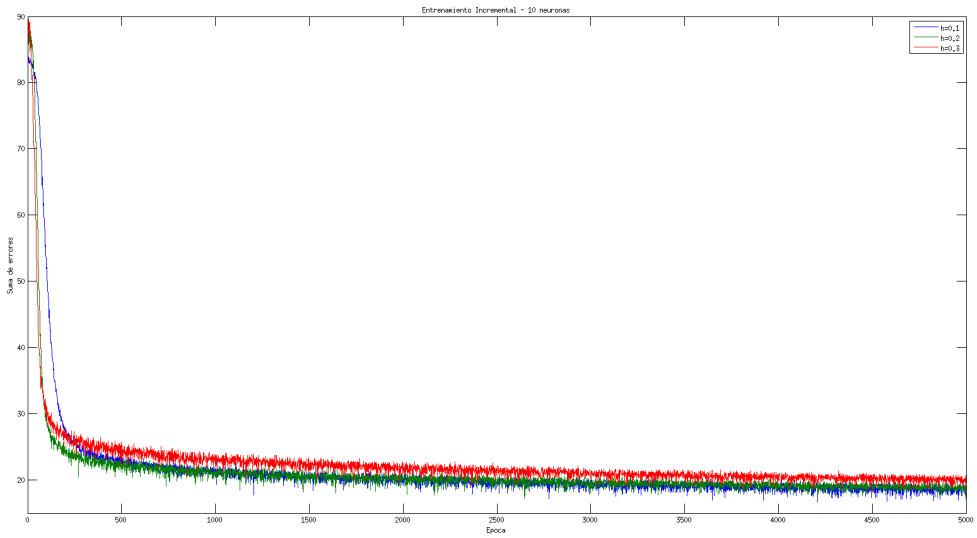

Figura 16: Suma de errores vs. Épocas - Entrenamiento incremental - 10 neuronas

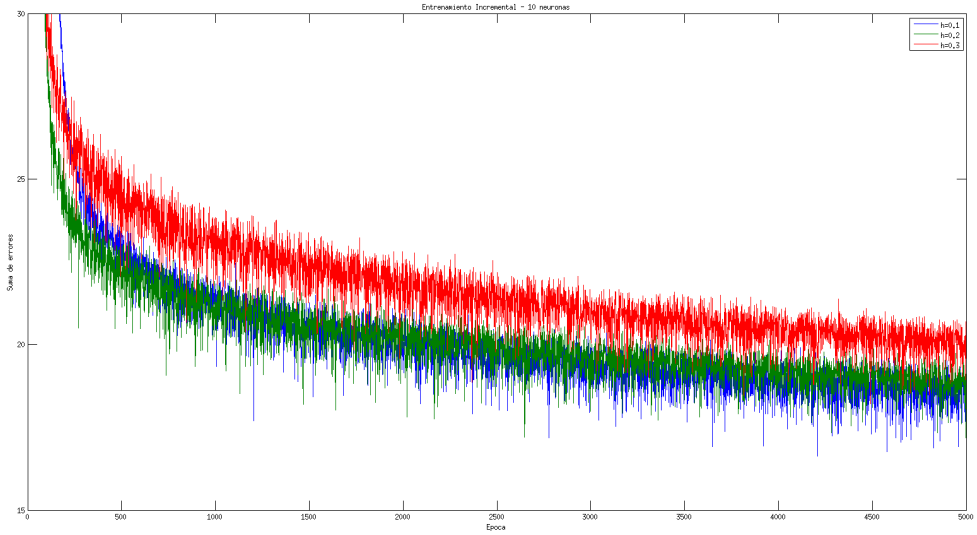

Figura 17: Suma de errores vs. Épocas - Entrenamiento incremental - 10 neuronas (Zoom)

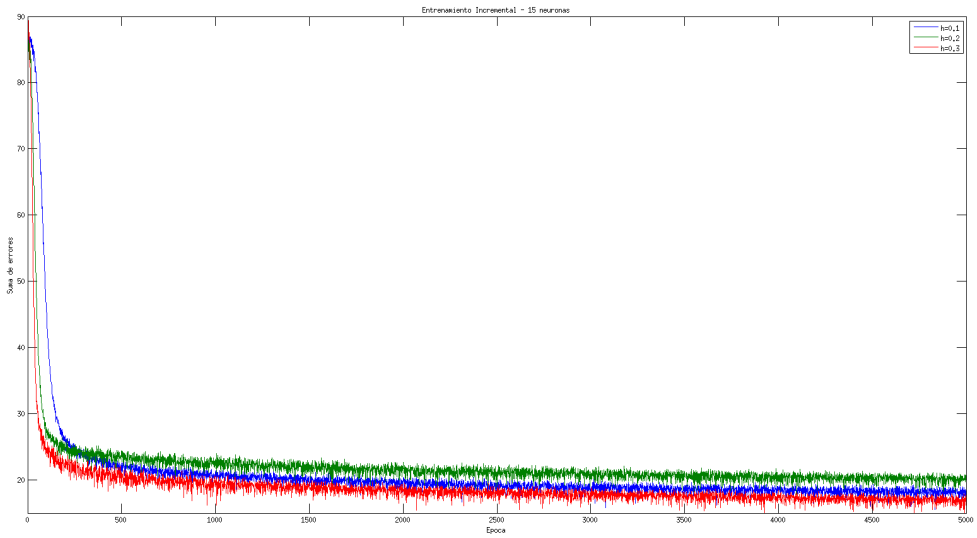

Figura 18: Suma de errores vs. Épocas - Entrenamiento incremental - 15 neuronas

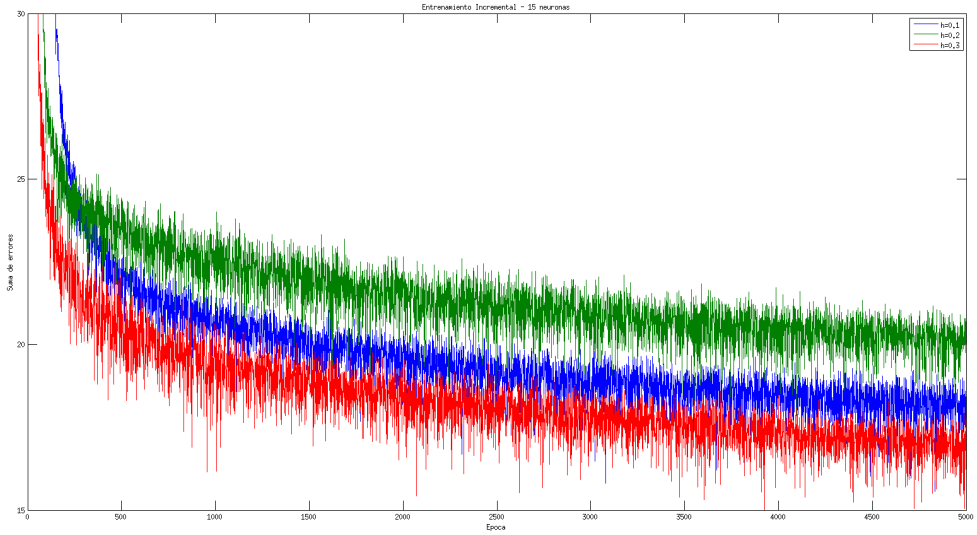

Figura 19: Suma de errores vs. Épocas - Entrenamiento incremental - 15 neuronas (Zoom)

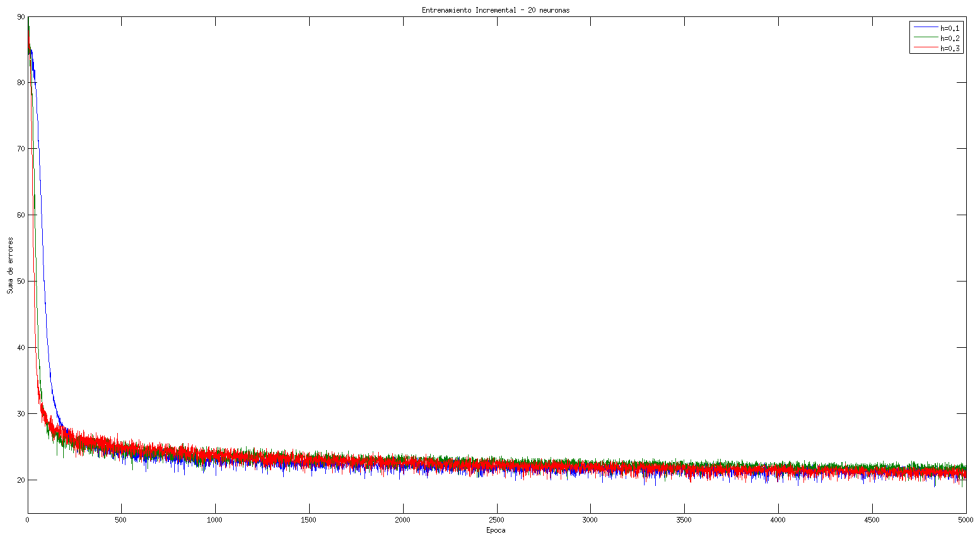

Figura 20: Suma de errores vs. Épocas - Entrenamiento incremental - 20 neuronas

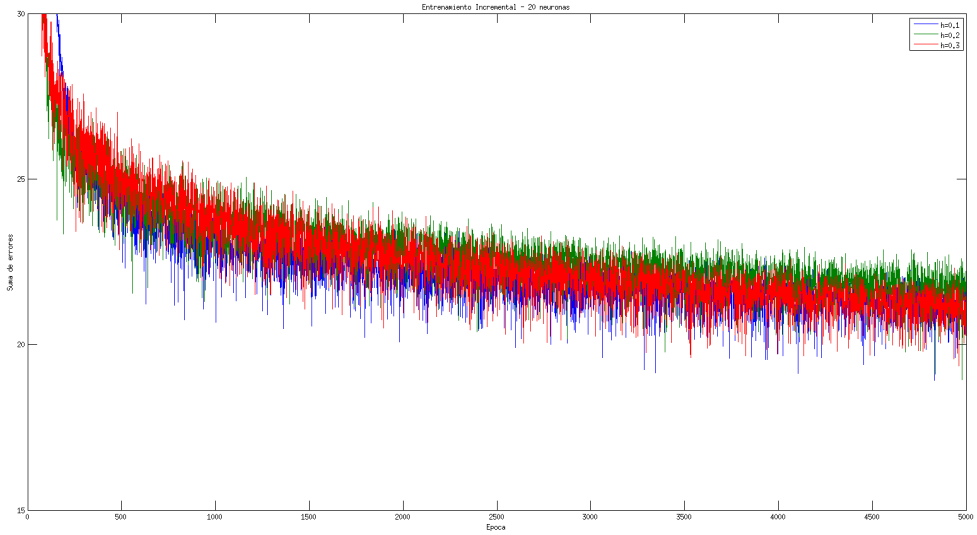

Figura 21: Suma de errores vs. Épocas - Entrenamiento incremental - 20 neuronas (Zoom)

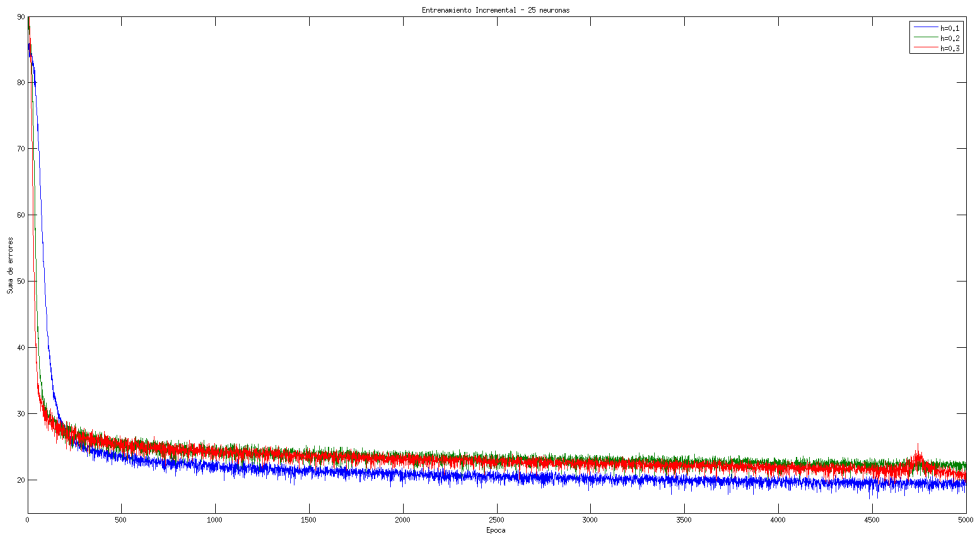

Figura 22: Suma de errores vs. Épocas - Entrenamiento incremental - 25 neuronas

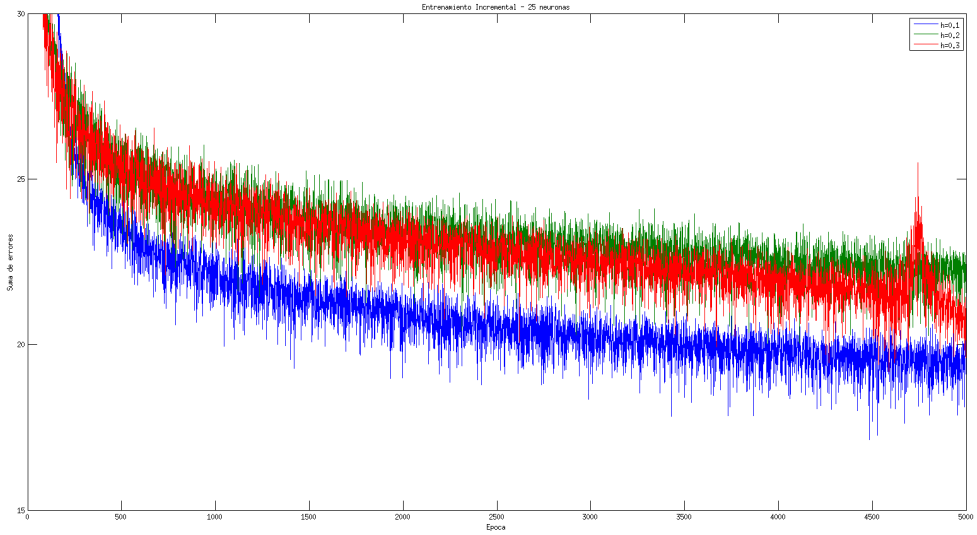

Figura 23: Suma de errores vs. Épocas - Entrenamiento incremental - 25 neuronas (Zoom)

Nuevamente como mencionamos anteriormente, no incluimos nuevamente gráficos de tiempo de estas corridas, dado que dichos tiempos se mantuvieron muy similares a los que mostramos antes. Es decir, como dijimos, los tiempos de cómputo parecen depender más que nada de la cantidad de épocas, el modo de aprendizaje y a lo sumo la cantidad de neuronas.

Mantuvimos en estos gráficos las mediciones hechas con $\eta = 0{,}1$ en el primer experimento para poder comparar. Podemos ver que todas las series dan resultados bastante parecidos de todas formas. A su vez, no hay una tendencia muy marcada sobre si algún valor de η fue el que obtuvo mejor error con todas las cantidad de neuronas, más bien hay casos donde un valor fue mejor que otro, y viceversa, parecería que más bien es una cuestión dependiente de la aleatoriedad de las corridas. Sin embargo son todas las series muy parecidas, y los valores que abarcan no difieren demasiado. No hay demasiado que podamos concluir de este experimento.

3.2. Conclusiones

De las corridas que realizamos con el modo de aprendizaje experimental vimos que no hubo una gran diferencia al usar los distintos valores de η propuestos. Sin embargo, el tiempo de cómputo requerido para el entrenamiento sí es muy sensible a la cantidad de neuronas presentes en la red, aumentando en gran medida. Algo que no mencionamos en el análisis previo, es que en este modo la *convergencia es bastante rápida*. Viendo los gráficos, podemos apreciar que en todas estas corridas que el mayor decrecimiento del error (la mayor mejora que se logra) se logra en dentro de las primeras 500 épocas (o antes incluso). Esto se corresponde con la base teórica que conocemos sobre este modo de aprendizaje. A partir de ahí, el error sigue decreciendo pero demasiado lentamente, de modo que uno tendría que sacrificar demasiado tiempo de cómputo para lograr mejores muy mínimas, lo cual no vale mucho la pena.

De las corridas usando el modo de batch vimos que se pueden conseguir resultados muy buenos con mucho menos tiempo de cómputo, y a la vez es un método mucho más sensible al valor de η elegido, y requirió una previa búsqueda de un valor que sea adecuado para obtener buenos resultados, mientras que con incremental los tres valores funcionaron igual de bien sin mucho problema. La curva de convergencia es un poco más lenta (hablando de épocas, no de tiempo) y al igual que en incremental, llegado un punto la curva se *plancha* por así decirlo.

Configuraciones recomendadas

Finalmente, mencionamos ahora cuáles son las configuraciones que nos parecen más convenientes para el entrenamiento de las redes con este dataset, tratando de balancear un buen resultado con un buen tiempo de cómputo. Otro detalle general sobre todos los resultados, es que parecería ser bastante difícil lograr que la suma de los errores baje más de aproximadamente 20, problamente esto esté relacionado con la forma del dataset combinado con las implementaciones de nuestros métodos. Sin embargo, esto puede verse de los resultados expuestos, no queriendo decir que no se puedan lograr errores menores, pero éstos requerirán una gran cantidad de cómputo extra. Podríamos decir que con este número tenemos una aproximación a una cota de error de aprendizaje.

Para esto, si se usa el modo incremental de aprendizaje, la cantidad de neuronas debe mantenerse baja, de lo contrario el tiempo requerido se vuelve muy elevado, y de todas formas la diferencia obtenida con cantidades mayores es muy poca. Dicho esto, recomendamos un valor de η cercano a 0.3, que fue el que mostró mejor desempeño con pocas neuronas. Si queremos que el entrenamiento logre aproximadamente el error deseado de 20 mencionado recién, se necesitan cerca de 500 épocas de entrenamiento, lo cual demora aproximadamente unos 2 minutos.

En cambio, usando el modo de aprendizaje batch, creemos que los mejores resultados se obtienen usando un valor de $\eta = 0{,}01$ o similar, y con el cual los mejores resultados se obtuvieron usando 25 neuronas, lo cual no afecta el tiempo de cómputo prácticamente. Elegimos este valor además porque en todas las corridas dio muy buenos resultados y además que su oscilación es bastante menor a la de los otros valores. Nuevamente, si queremos lograr un error deseado aproximado de 20 como mencionamos, necesitaremos con este modo cerca de 3000 épocas, lo cual necesita más o menos unos 3 minutos.

Finalmente, por la robusteza que tiene frente a diferentes valores de η, lo cual hace que su uso más simple, para lograr un error de aproximadamente 20, creemos que es más conveniente usar el modo incremental, además por ser de rápida convergencia y ser más rápido.

Si por otro lado, lo que se busca es reducir el error cuanto sea posible, la combinación usando modo batch permite bajar el error (aunque lentamente) más rápido y de una manera más pareja.

4. Experimentación sobre el dataset de *Eficiencia energética*

Para esta segunda experimentación volveremos a utilizar parámetros similares a la primera. Si bien ya pudimos contemplar algunas características que generan los parámetros para el caso particular de cualquier dataset, volveremos a realizar el mismo análisis para profundizar sobre estos datos.

En principio utilizaremos nuevamente una capa oculta, pero esta vez con 5, 10 y 15 nodos por los altos tiempos de cómputo que demanda realizar 5 corridas de cada caso. Creemos que tendrá que haber una diferencia sustancial al utilizar alguno de estos parámetros dado que en el experimento anterior se tenían 30 neuronas de entrada y 1 de salida; mientras que ahora hay 8 de entrada y 2 de salida y la cantidad de conexiones se maneja de forma diferente.

Nuevamente veremos qué sucede tanto para los esquemas *batch* e *incremental* y combinaremos éstas pruebas con variar la función sigmoidea. Como valores de aprendizaje utilizaremos $\eta \in \{0{,}05, 0{,}1, 0{,}2, 0{,}3, 0{,}4, 0{,}5\}$ ya que permite obtener un amplio panorama. Para la cantidad de épocas máxima utilizaremos el valor 10000 con un error mínimo de 0.0001 para forzar que los entrenamientos se ejecuten hasta el final.

Por último el porcentaje del dataset destinado a entrenamiento será del $80\,\%$, ya que nos parece una buena cantidad para aprender y $20\,\%$ deja mucho para testear.

Además de todo esto, pensamos que sería conveniente probar con algunos casos que escapen un poco a lo que venimos viendo. Es por esto que al final de la prueba que acabamos de detallar, realizaremos tres pruebas más:

- Veremos qué sucede al utilizar los métodos batch e incremental con $\eta \in \{0{,}01, 0{,}001, 0{,}0001\}$, usando 5 nodos en la capa oculta.
- Veremos qué sucede al utilizar menor cantidad de nodos en la capa interna, en particular con 2, 3 y 4 utilizando $\eta = 0{,}0001$, la función *bipolar* y el método batch.
- Veremos qué sucede al usar mayor cantidad de nodos en la capa interna, como ser 30 utilizando $\eta = 0{,}0001$, la función *bipolar* y el método batch.

4.1. Resultados y Análisis

Primera etapa de experimentación

Como primera prueba veremos qué sucede al utilizar una capa interna con 5 nodos variando los valores de η entre $\{0{,}05, 0{,}1, 0{,}2, 0{,}3, 0{,}4, 0{,}5\}$. Al igual que con el anterior dataset, siempre elegimos entrenar con un $80\,\%$ de los elementos, usando un $20\,\%$ para testing, por lo cual dado que los mismos se eligen siempre de forma uniformemente aleatoria y realizamos las corridas 5 veces para evitar outliers nos interesará conseguir el error de entrenamiento más bajo. Recordemos que para estas pruebas siempre utilizamos 10000 épocas de aprendizaje para poder analizar qué sucede en cada cantidad de épocas.

Veamos qué resultó utilizando el método de aprendizaje incremental usando la sigmoidea *binomial*.

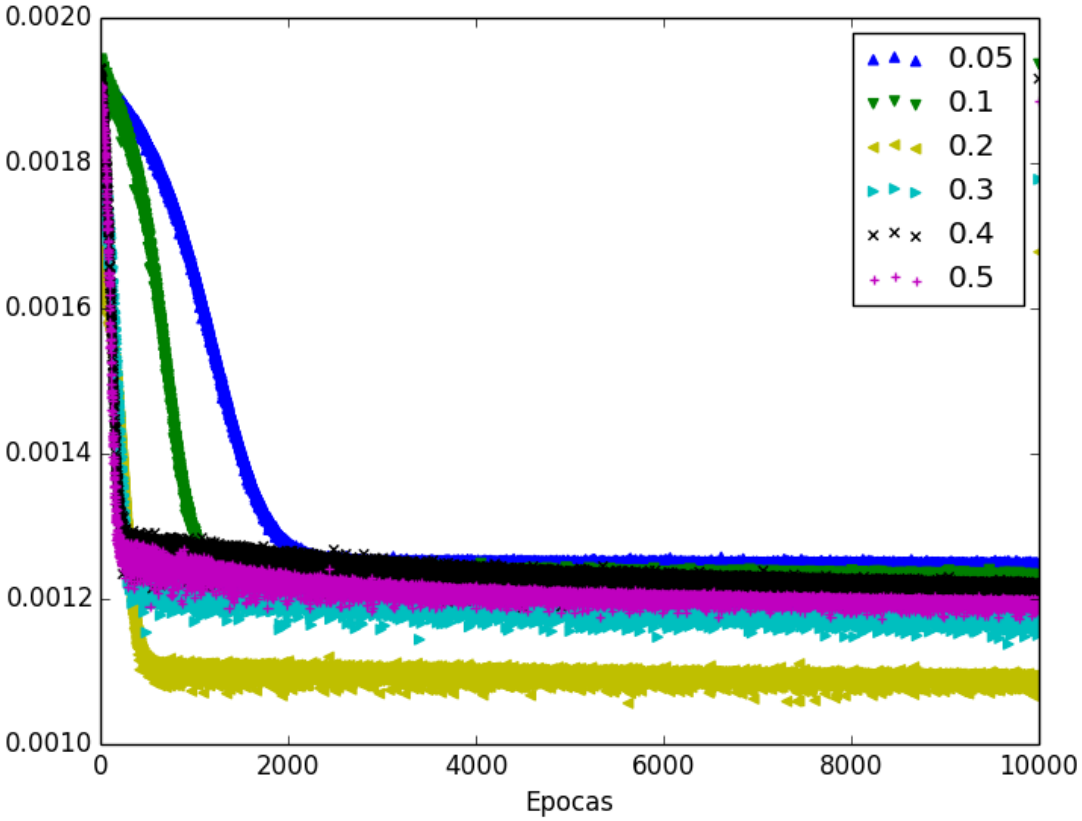

Figura 24: Error promedio en función del valor de eta - Binomial con método incremental

Podemos ver que los errores resultaron ser muy bajos y aceptables desde las pocas épocas. Tal es así que, si bien no se estancó y siempre siguió bajando, las diferencias de error al final eran del orden de 10^{-3}.

Veamos qué sucedió con el método de aprendizaje batch usando los mismos parámetros.

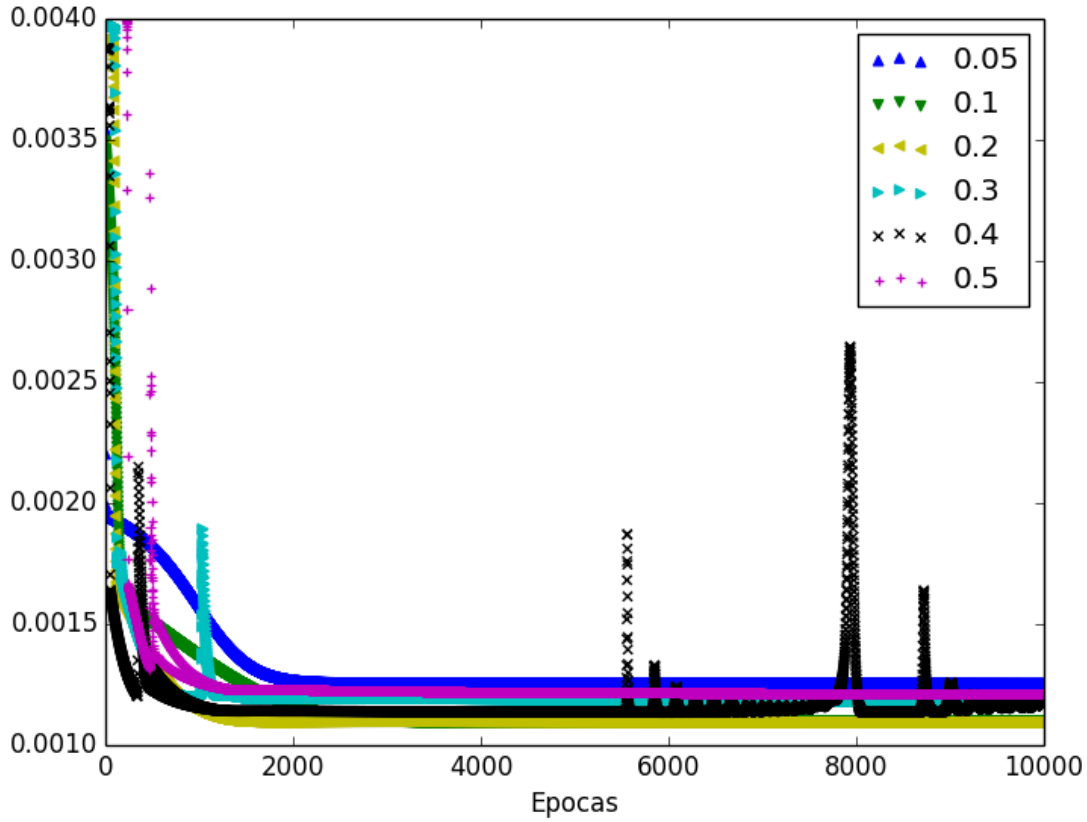

Figura 25: Error promedio en función del valor de eta - Binomial con método batch

En este caso podemos ver que también se consiguió un nivel de error similar por lo cual en principio estos dos métodos no consiguen mayores diferencias por ahora. Por otro lado, sí es cierto que a la red esta vez le costó menos conseguir un valor pequeño de error (observar que en el gráfico anterior, cuando se usaba $\eta = 0{,}1$ o $\eta = 0{,}05$ se necesitaban hasta 2000 épocas para conseguir lo mismo que acá demandaba menor cantidad de épocas. También hay que resaltar que en este segundo caso se presentaron algunos saltos esporádicos en los errores y esto, creemos, es debido a la forma en la que se actualizan al final los datos al utilizar el método batch.

Para ver esto más en detalle podemos observar la siguiente tabla de diferencias cuando se consiguen las 10000 épocas de entrenamiento.

Modo	0,05	0,1	0,2	0,3	0,4	0,5
Batch	0,001252912	0,001096400	0,001091977	0,001181271	0,001159109	0,001214512
Incremental	0,001151398	0,001161981	0,001141313	0,001179913	0,001216006	0,001229815

Cuadro 1: Tabla de error promedio en función del valor de eta - Binomial

Además consideramos la suma de la diferencia de los errores al cuadrado, en la siguiente tabla.

Modo	0,05	0,1	0,2	0,3	0,4	0,5
Batch	0,501164995	0,438560079	0,436791010	0,472508503	0,463643868	0,485804883
Incremental	0,477362727	0,492307919	0,436351098	0,468716319	0,487700431	0,478032797

Cuadro 2: Tabla de suma de diferencias de errores en función del valor de eta - Binomial

Lo que nos dice que el error a encontrar en cualquier caso de entrenamiento será muy pequeño y en particular similar, pasando las 2000 épocas (que es lo que se podía ver en el gráfico).

Veamos a continuación qué sucede al mirar los errores sobre la validación.

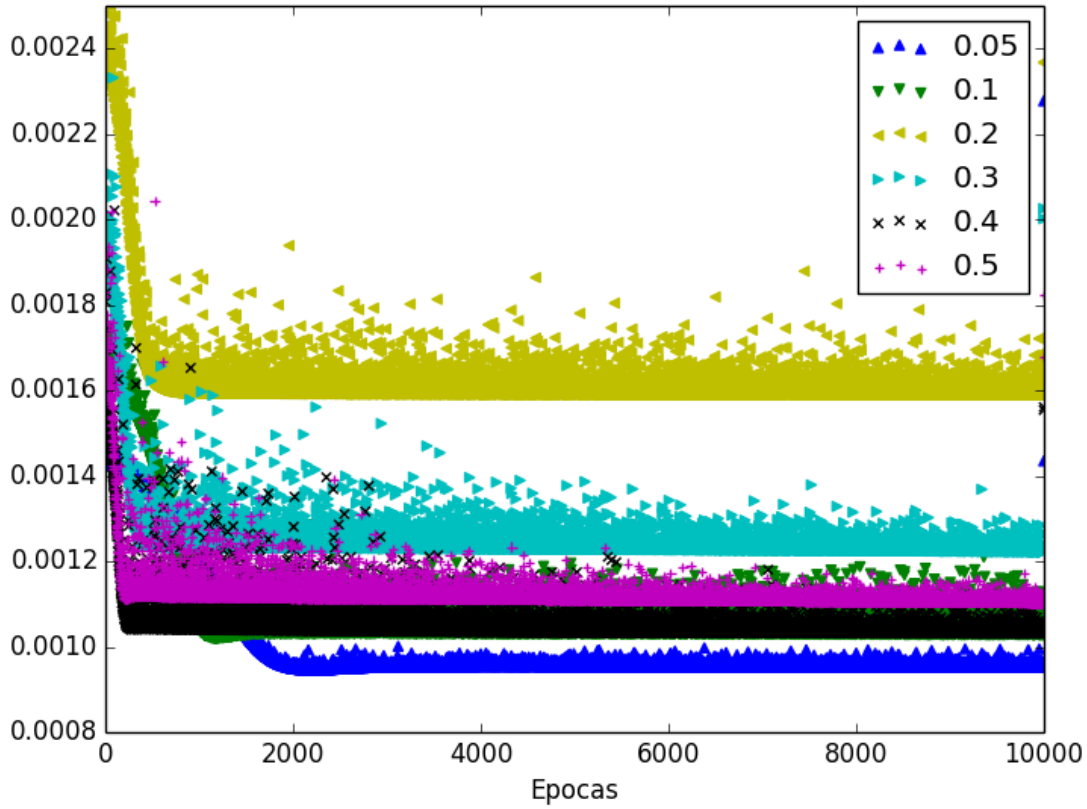

Figura 26: Error promedio en función del valor de eta - Binomial con método incremental

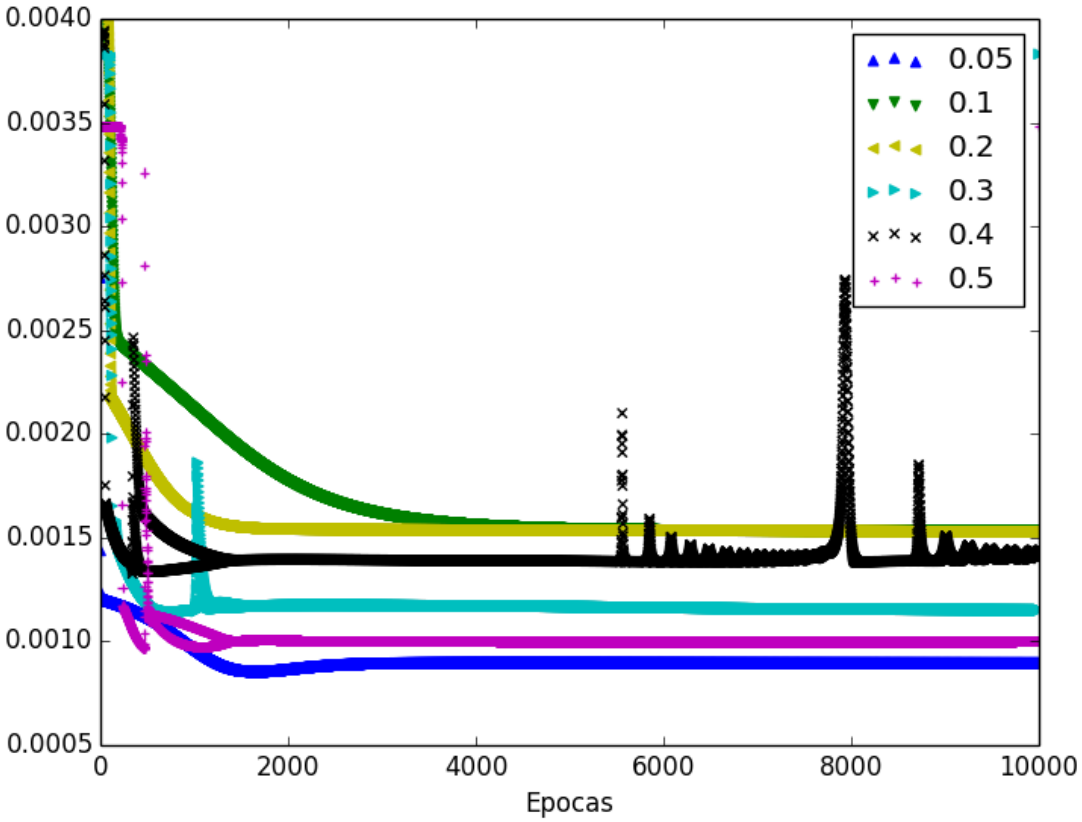

Figura 27: Error promedio en función del valor de eta - Binomial con método batch

En la primera figura podemos ver que el error promedio de testing presenta una tendencia a bajar a medida que aumenta la cantidad de épocas máxima. Además, si bien se trata de errores muy pequeños desde las 100 épocas en adelante, podemos ver que a partir de las 2000 (usando $\eta = 0{,}05$) conseguimos errores mucho más pequeños por lo que utilizar este valor de η termina conveniendo a nuestro criterio.

Veamos ahora qué sucedió al utilizar la otra función sigmoidea, es decir la *bipolar*. Realizamos el mismo análisis que antes partiendo del método incremental.

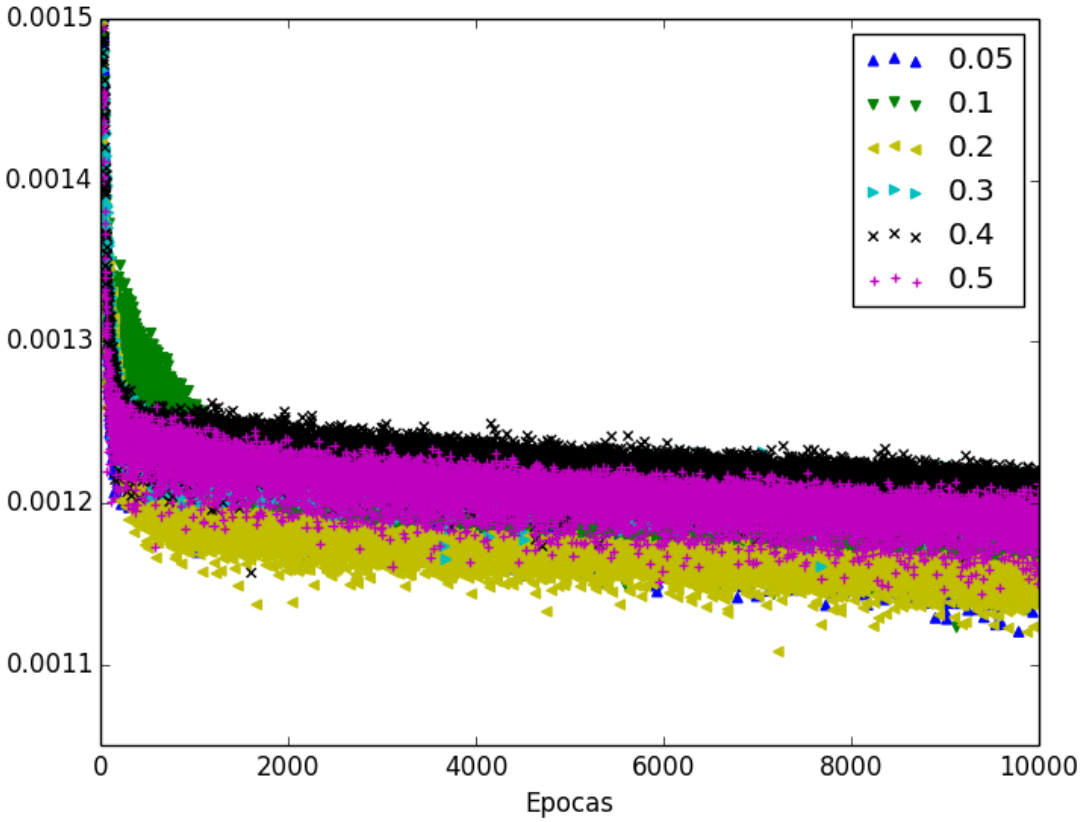

Figura 28: Error promedio en función del valor de eta - Bipolar con método incremental

Se puede ver que obtuvimos unos valores dentro de todo parecidos a la prueba anterior con la *binomial*.

Veamos qué sucedió con el método de aprendizaje batch usando los mismos parámetros.

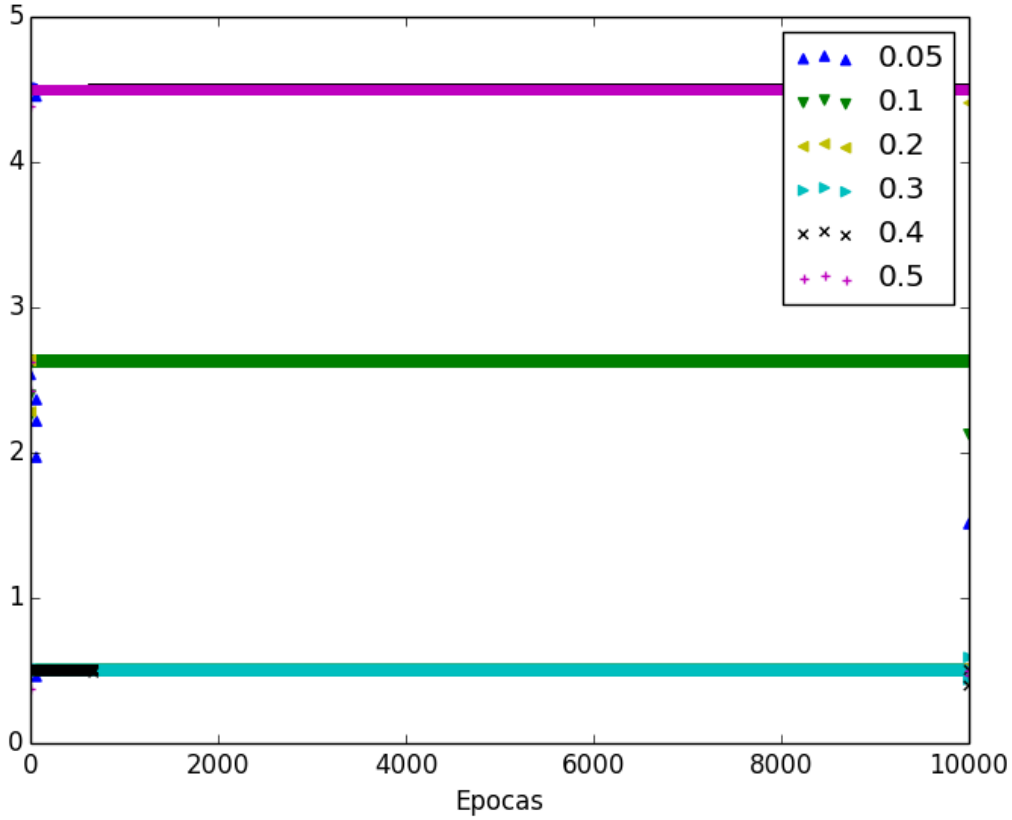

Figura 29: Error promedio en función del valor de eta - Bipolar con método batch

Podemos ver que en este caso los errores se dispararon y no se alcanzó reducirlos a un número aceptable ni siquiera en las 10000 épocas. El error bajó hasta que en un momento, pasadas las 50 épocas, se estancó. Por lo cual nos hace empezar a ver que la función *bipolar* se podría comportar muy mal para este dataset al utilizar el método batch, al menos para los presentes parámetros.

Para mayor detalle, volvamos a ver las diferencias conseguidas en estas pruebas.

Modo	0,05	0,1	0,2	0,3	0,4	0,5
Batch	0,503900982	2,630075912	0,504226899	0,503922903	4,503810320	4,503773001
Incremental	0,001151398	0,001173624	0,001154999	0,001207040	0,001209739	0,001170256

Cuadro 3: Tabla de error promedio en función del valor de eta - Bipolar

Además consideramos la suma de la diferencia de los errores al cuadrado, en la siguiente tabla.

Modo	0,05	0,1	0,2	0,3	0,4	0,5
Batch	201,5603931	1052,030364	201,6907596	201,5691612	1801,524128	1801,509200
Incremental	0,461999726	0,469449854	0,460571850	0,482816347	0,483895924	0,468102437

Cuadro 4: Tabla de suma de diferencias de errores en función del valor de eta - Bipolar

Lo que nos dice que, al menos para estos parámetros, la función *bipolar* con método batch se comporta muy mal, mientras que con incremental se vuelve prácticamente equivalente a utilizar la sigmoidea

binomial.

Veamos qué sucedió esta vez al ver la validación.

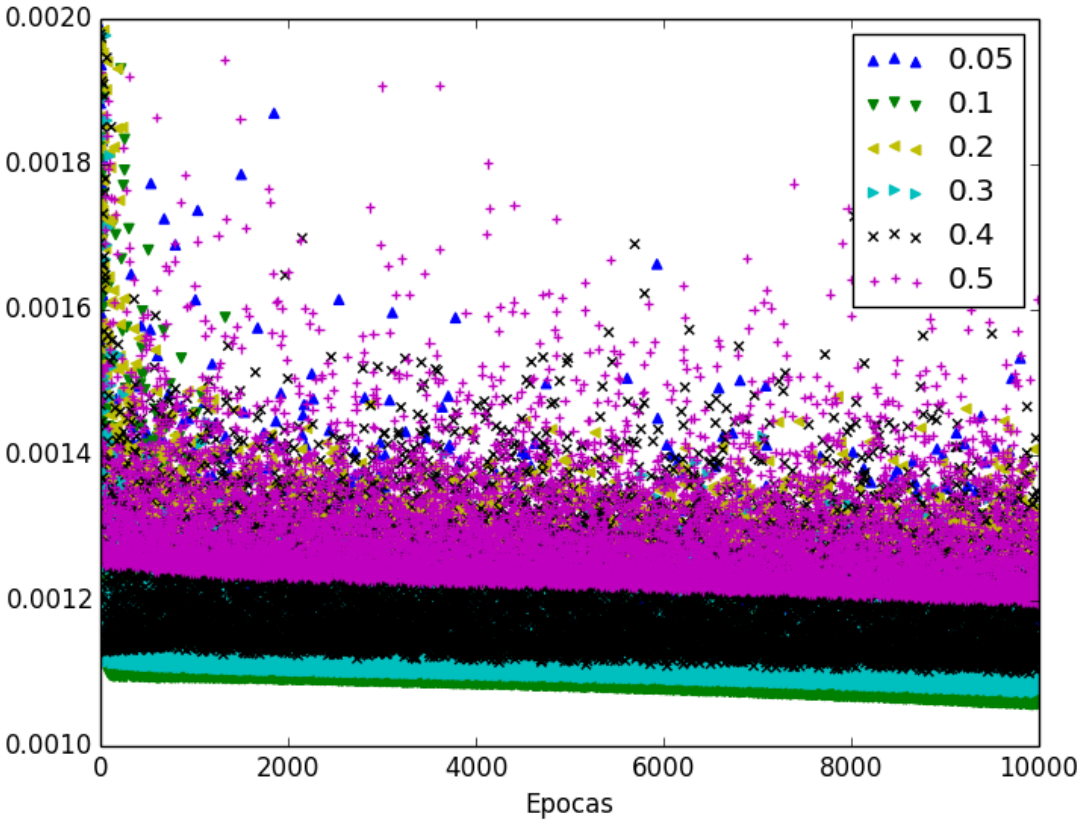

Figura 30: Error promedio en función del valor de eta - Bipolar con método incremental

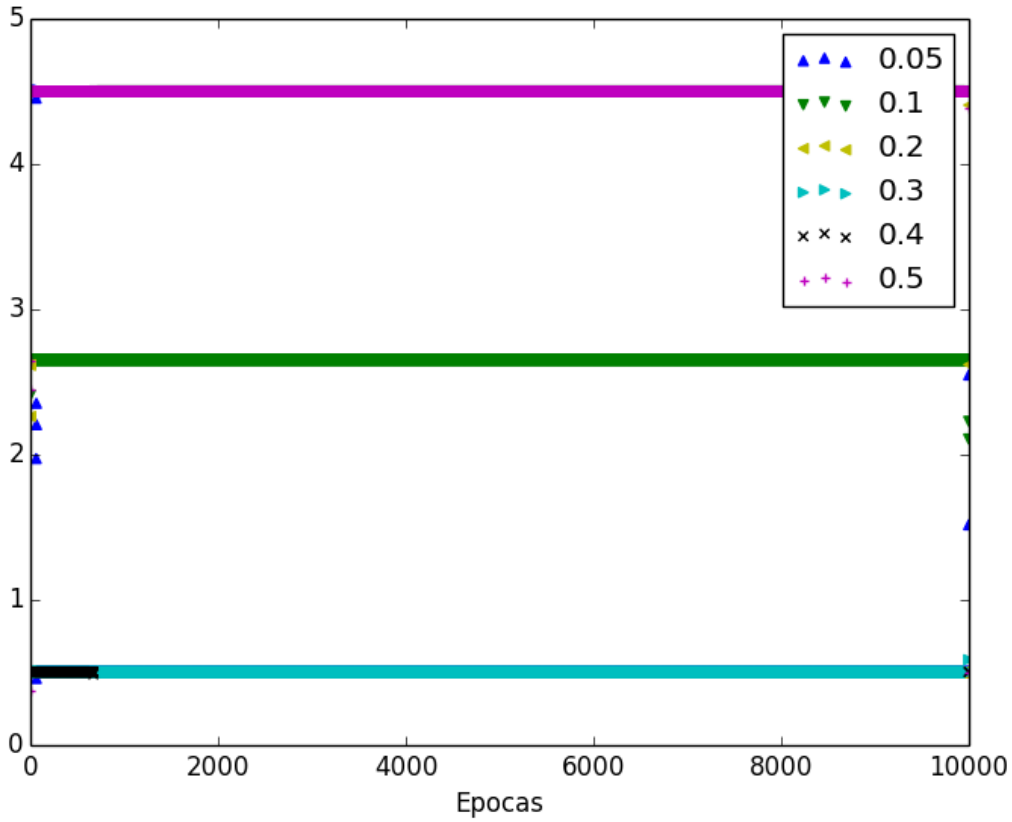

Figura 31: Error promedio en función del valor de eta - Bipolar con método batch

En la primera figura podemos ver que el error promedio de testing presenta una tendencia a bajar a medida que aumenta la cantidad de épocas máxima. Además, comienza a ser muy bajo a partir de las 100 épocas por lo cual esta combinación de parámetros se convierte en una opción prometedora.

Por otro lado, vemos en el testing que al utilizar método batch los errores son prácticamente tan altos como lo son al entrenar, por lo cual esta combinación de parámetros son, irónicamente, un error.

Veamos como siguiente prueba lo sucedido al utilizar 10 nodos en la capa interna.

En principio y para no volver a presentar gráficos que son prácticamente iguales, notamos que con la función *binomial* los resultados fueron casi exactamente los mismos. Esto nos hizo ver que (si bien lo analizaremos mejor más adelante) esta función tiende a soportar incrementos de neuronas en la capa interna sin variar el error. Es por esto que decidimos mostrar a continuación el caso de la *bipolar* que nos parecía más interesante. El orden de los gráficos es por filas, de izquierda a derecha de arriba a abajo, en donde se prueba en cada uno los distintos valores de η, comenzando con $0{,}05$ y terminando en $0{,}5$.

Primero mostramos qué sucedió con el método incremental. Utilizamos este nivel de zoom para no perder noción de cuánto valen las primeras épocas en relación al error que se consigue al final. De todas formas esto viene más adelante sustentado con tablas de valores.

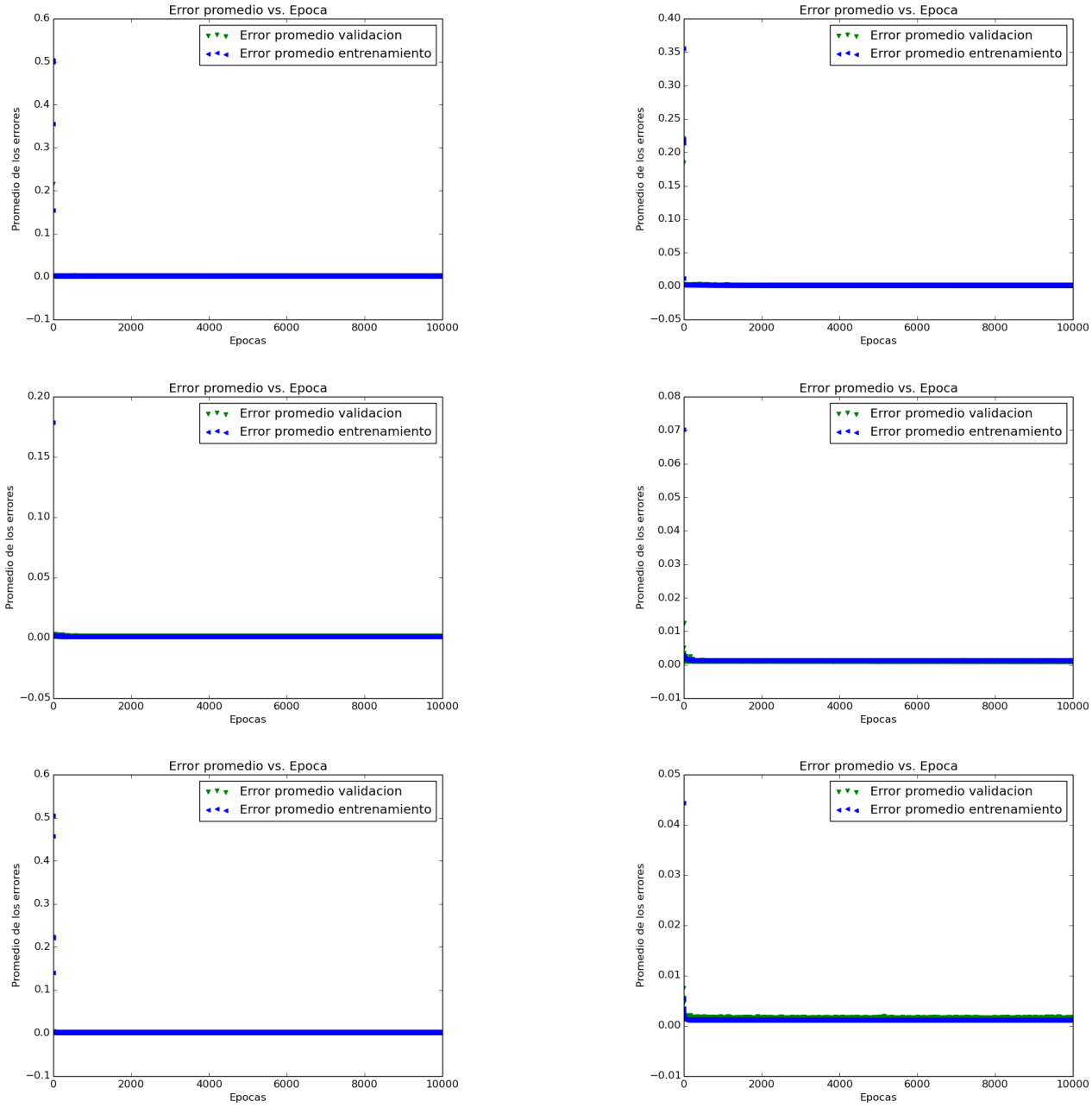

Figura 32: Errores en función de la época, para cada valor de η - Bipolar con método incremental

Luego qué sucedió con el método batch.

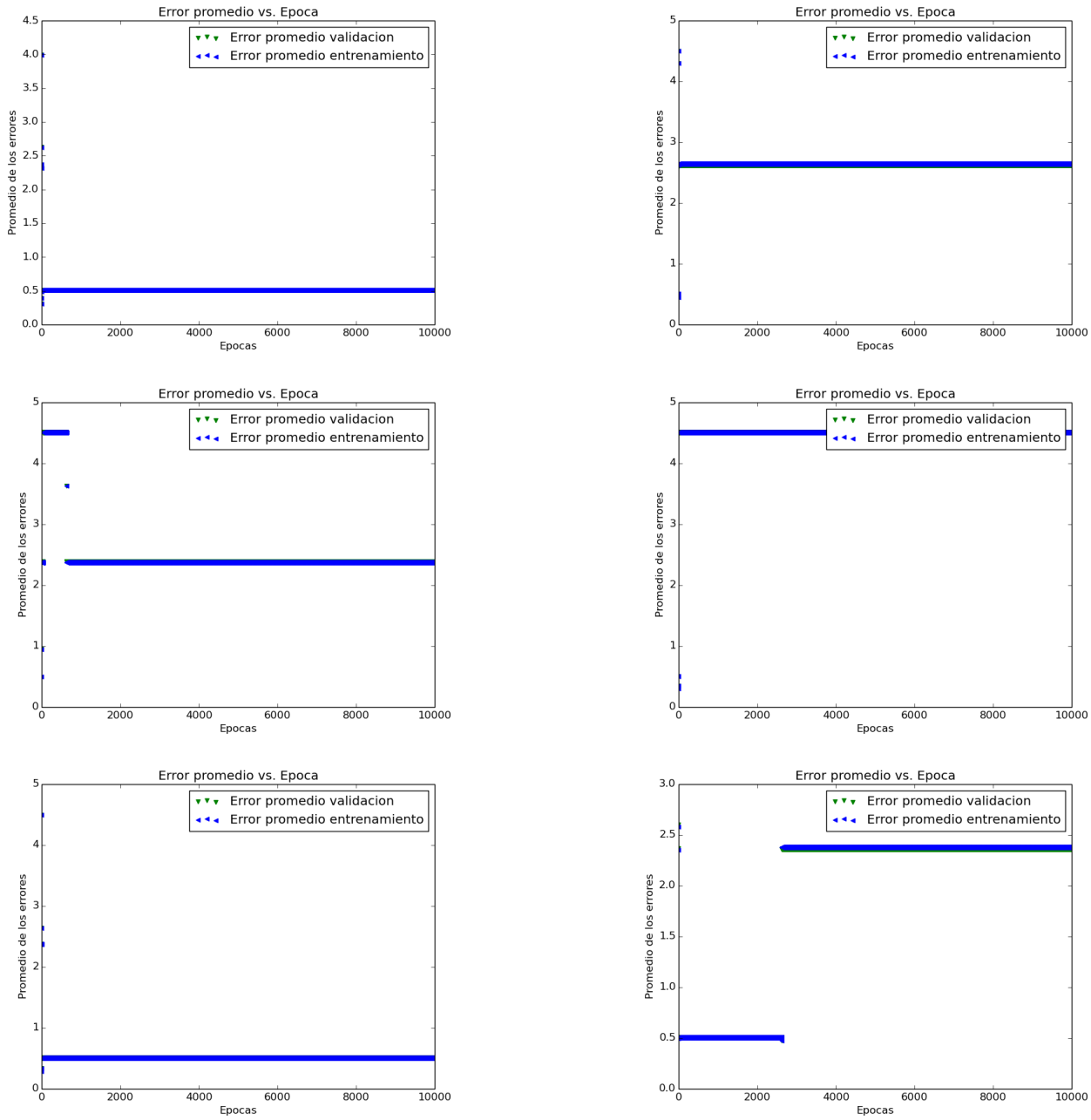

Figura 33: Errores en función de la época, para cada valor de η - Bipolar con método batch

Aca podemos ver que las tendencias son las mismas que hubo antes: el método incremental presentó un muy buen rendimiento con respecto al error mientras que el batch arrojaba valores muy elevados del mismo.

Además volvemos a ver la misma propiedad que presentó el batch con 5 nodos (en la última prueba de esa sección) en donde se acumula en un valor de error y de un momento a otro el error se dispara. La hipótesis que tenemos acerca de este comportamiento es la misma que dijimos antes: los valores se realizan por batch y un pequeño cambio en una época hace que se realice todo junto e impacte de mayor forma.

Para ver más detalladamente lo que muestran estos gráficos, vemos el resultado obtenido en la época 10000.

$Modo$	0,05	0,1	0,2	0,3	0,4	0,5
$Batch$	0,503772748	2,63930736	2,37080875	4,50398717	0,503906846	2,37739854
$Incremental$	0,001136507	0,001129846	0,00111615	0,001234683	0,001215086	0,001132203

Cuadro 5: Tabla de error promedio en función del valor de eta - Bipolar

Y como suma de errores se obtuvo lo siguiente.

$Modo$	0,05	0,1	0,2	0,3	0,4	0,5
$Batch$	201,5090994	1055,72294	948,3235021	1801,594868	201,5627386	950,9594172
$Incremental$	0,454602969	0,451938649	0,446462619	0,493873541	0,486034496	0,452881248

Cuadro 6: Tabla de suma de diferencias de errores en función del valor de eta - Bipolar

Esto nos muestra la gran diferencia de errores obtenidos con batch e incremental. Lo que, para esta combinación de parámetros, nos hace elegir sin dudas el método incremental.

Por último, en la validación se comportó de forma muy similar a la anterior y respetó la misma tendencia de gráficos que hubo antes para la función bipolar.

Por último terminamos esta primera etapa con una tercera prueba utilizando una cantidad de 15 neuronas en la capa interna. Los resultados fueron nuevamente iguales con la función *binomial*. Por lo cual con este caso podemos ver que, al menos en lo que a nuestras pruebas se refiere, la función *binomial* se comporta muy bien y de igual manera para capas intermedias de entre 5 y 15 nodos.

Mostramos por último cómo se vieron los resultados al utilizar la bipolar.

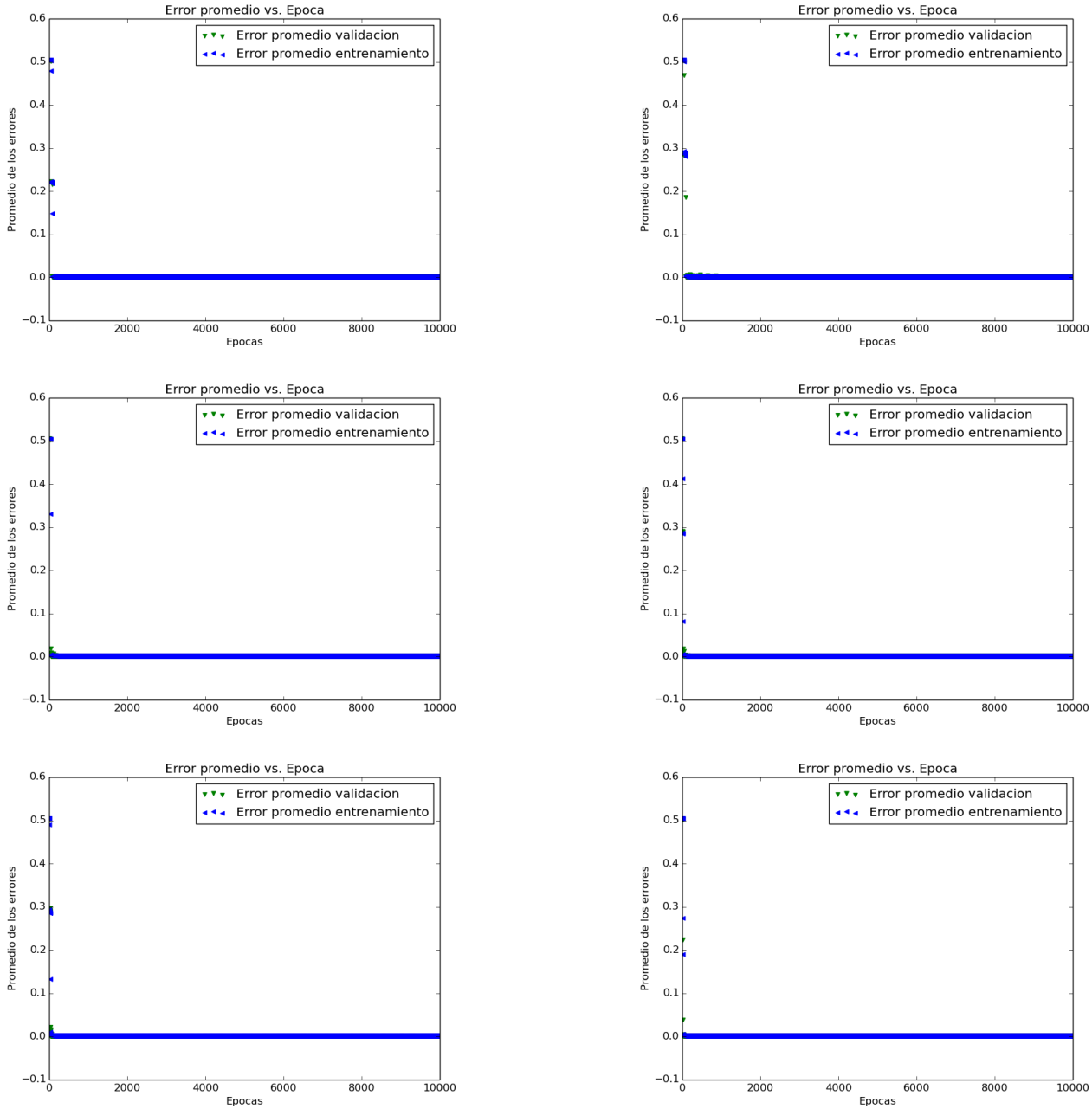

Figura 34: Errores en función de la época, para cada valor de η - Bipolar con método incremental

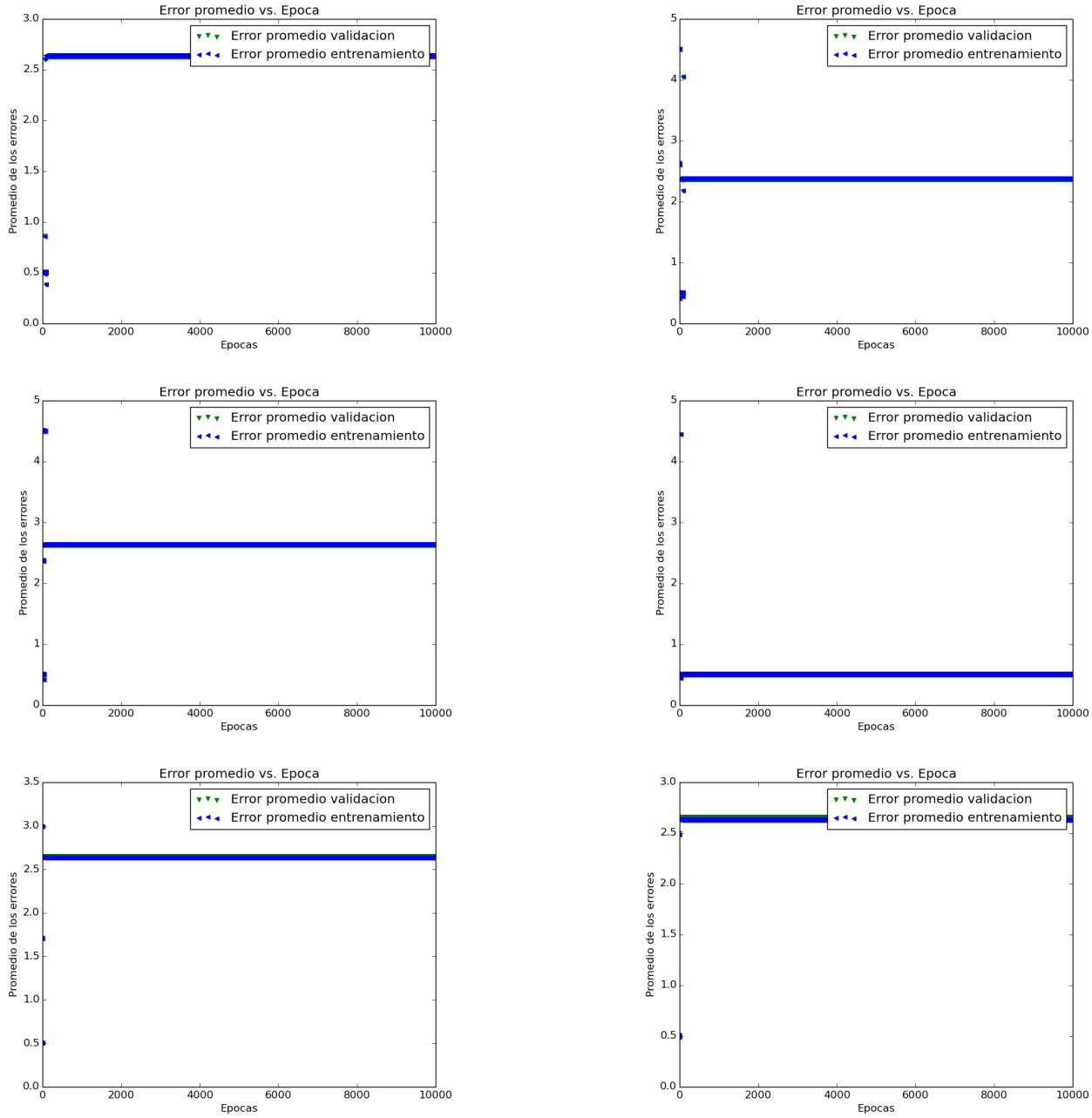

Figura 35: Errores en función de la época, para cada valor de η - Bipolar con método batch

En estas figuras podemos nuevamente ver el buen rendimiento del método incremental y el elevado error que acarrea utilizar batch para esta sigmoidea.

Para ver más detalladamente lo que muestran estos gráficos, vemos el resultado obtenido en la época 10000.

$Modo$	0,05	0,1	0,2	0,3	0,4	0,5
$Batch$	0,503772748	2,63930736	2,37080875	4,50398717	0,503906846	2,37739854
$Incremental$	0,001136507	0,001129846	0,00111615	0,001234683	0,001215086	0,001132203

Cuadro 7: Tabla de error promedio en función del valor de eta - Bipolar

Y como suma de errores se obtuvo lo siguiente.

$Modo$	0,05	0,1	0,2	0,3	0,4	0,5
$Batch$	201,5090994	1055,72294	948,3235021	1801,594868	201,5627386	950,9594172
$Incremental$	0,454602969	0,451938649	0,446462619	0,493873541	0,486034496	0,452881248

Cuadro 8: Tabla de suma de diferencias de errores en función del valor de eta - Bipolar

Por último, en la validación se comportó muy similar al primer caso de 5 nodos por lo cual no agregamos gráficos para no ser reiterativos.

Como primera conclusión respecto de lo que pudimos ver, observamos que para redes con entre 5 y 15 nodos el rendimiento (en base al error) se mantuvo en muy buenos valores para la sigmoidea *binomial* usando ambos métodos. Por otro lado, la función *bipolar* se comportó de igual manera al utilizar el método incremental, aunque su rendimiento empeoró de forma desmedida cuando se utilizó el método batch. Por lo cual, creemos que esta última combinación no resulta favorable para este dataset. Además de esto, no se presentaron mayores variaciones de errores al tomar distintos valores de η, al menos para la función *binomial*.

Veamos a continuación un gráfico que nos muestra las diferencias de tiempo al ejecutar los métodos batch e incremental para la sigmoidea *binomial*, según la cantidad de épocas. Para esto decidimos utilizar un valor de $\eta = 0{,}1$ al igual que realizamos para el anterior dataset.

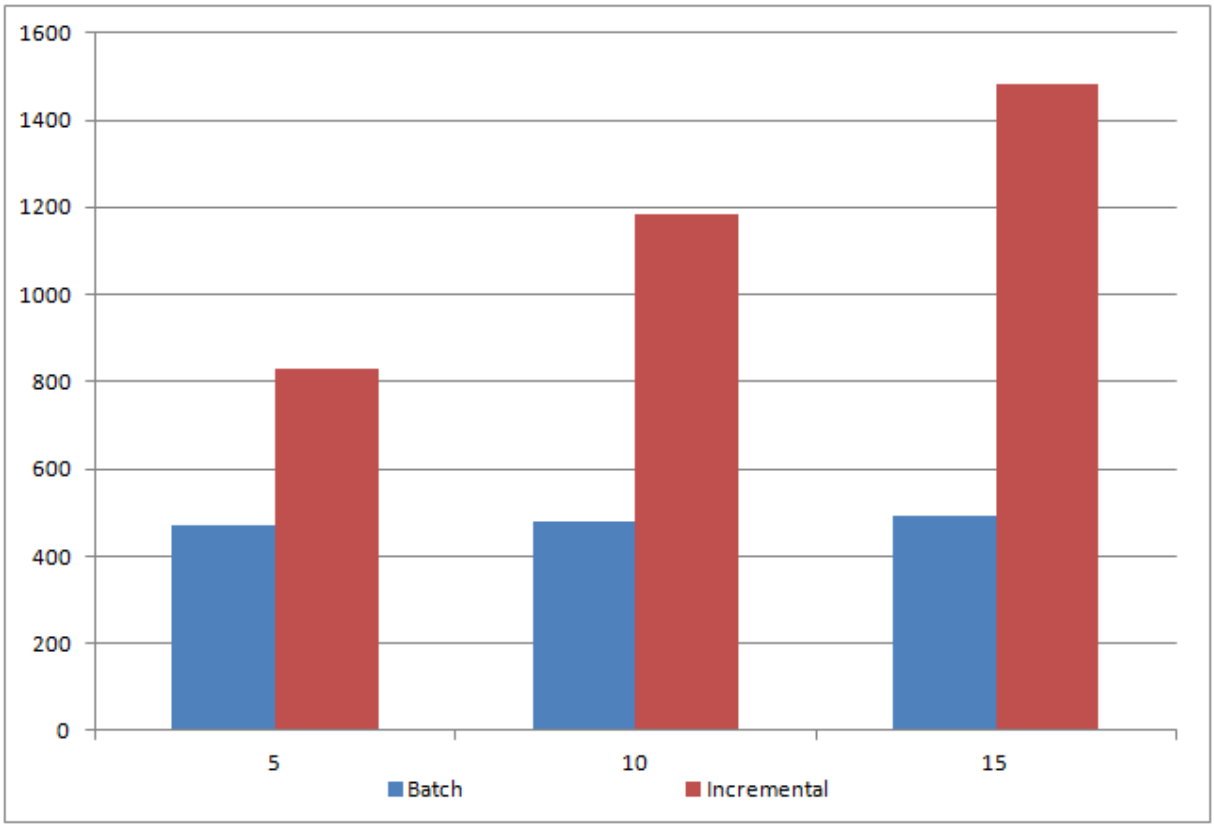

Figura 36: Tiempo promedio en función del método usado - Red con 5 neuronas internas y $\eta = 0{,}1$

Podemos ver que el método batch se vuelve mucho más ágil y menos demandante para mayor cantidad de épocas, por lo cual esto se vuelve un factor decisivo a la hora de elegir uno de estos dos métodos. Sabiendo que, como vimos antes, incremental y batch obtienen prácticamente los mismos resultados al utilizar la sigmoidea *binomial*, en la práctica notamos que termina conveniendo utilizar el método batch a partir de las 2000 épocas. Esto último se puede deducir por la figura 27. Además, entre los η por los que se puede optar, creemos que el valor $\eta = 0{,}05$ es el mejor para estos casos ya que obtiene un error promedio pequeño (del orden de 10^{-3}) a partir de las 10 épocas, por lo cual esta última elección sería la mejor desde el punto de vista de nuestra experimentación.

Segunda etapa de experimentación

Para esta segunda prueba analizaremos lo que sucede al utilizar el método batch y la sigmoidea *bipolar* con valores muy pequeños de η, entre ellos $\{0{,}01, 0{,}001, 0{,}0001\}$. Si bien las pruebas las realizamos con 10000 épocas, para mayor entendimiento de lo que sucede graficamos hasta la época 100. A continuación mostramos los resultados.

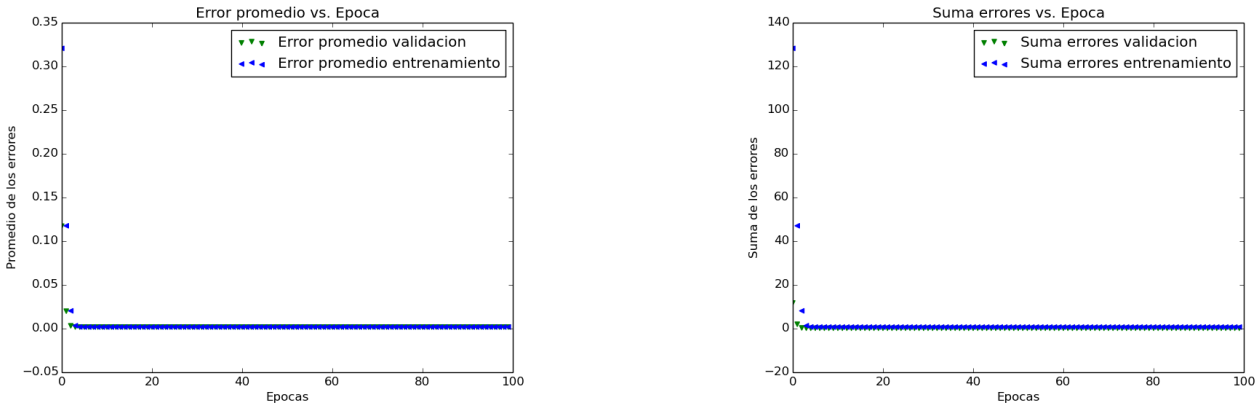

Figura 37: Errores en función de la época utilizando $\eta = 0{,}01$ - Bipolar con método batch

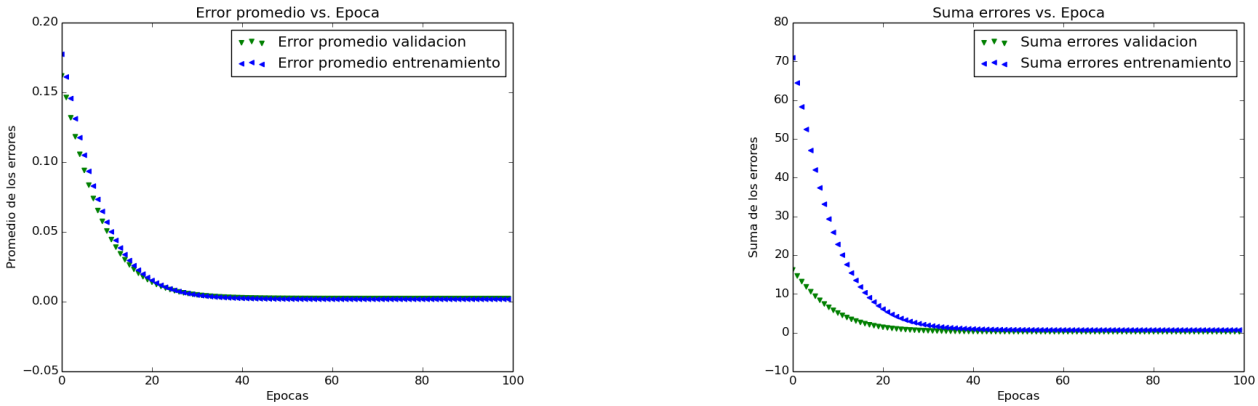

Figura 38: Errores en función de la época utilizando $\eta = 0{,}001$ - Bipolar con método batch

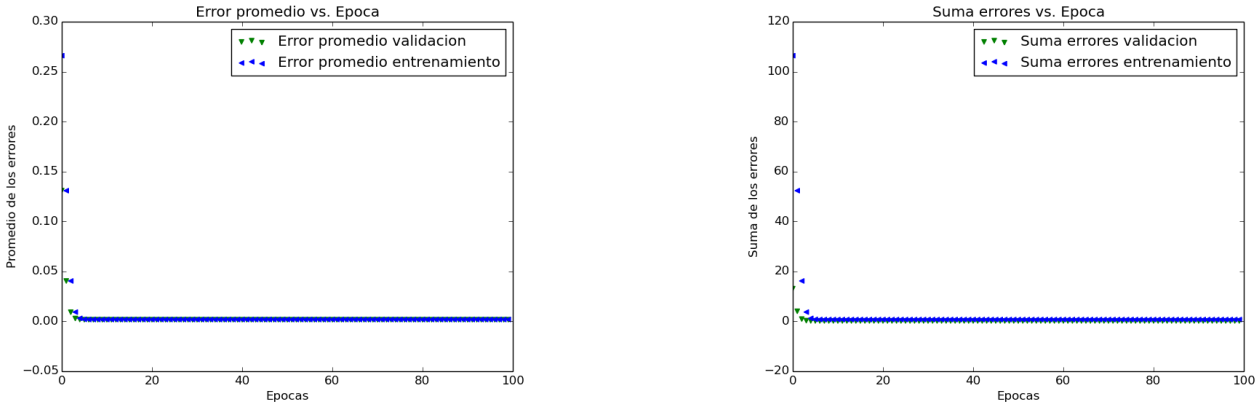

Figura 39: Errores en función de la época utilizando $\eta = 0{,}0001$ - Bipolar con método batch

Como se puede observar, un valor de $\eta = 0{,}01$ consiguió reducir el error muy tempranamente, para las 5 épocas. El error fue de 0.001182943 para dicha época y si bien seguía reduciéndose, lo hacía tan lentamente que para las 10000 épocas se volvía imperceptible.

Por otro lado, con $\eta = 0{,}001$ se consiguió reducir notablemente hasta la época 30, a partir de la cual los cambios se vuelven imperceptibles. Además, el error promedio conseguido en dicha época fue de 0.001249024, con lo que fue similar al caso anterior.

En el último caso se vuelve a ver lo mismo que con $\eta = 0{,}01$, por lo cual creemos que en la práctica se comportarían dentro de todo iguales.

Finalmente los errores de validación se comportaron de manera similar a los errores de entrenamiento, por lo cual creemos que si tenemos que elegir entre los valores de η probados, nos quedamos con $\eta = 0{,}01$ y $\eta = 0{,}0001$ como los mejores respecto de nuestra experimentación.

Tercera etapa de experimentación

Veamos ahora cómo resultó la tercera prueba al utilizar 2, 3 y 4 neuronas intermedias.

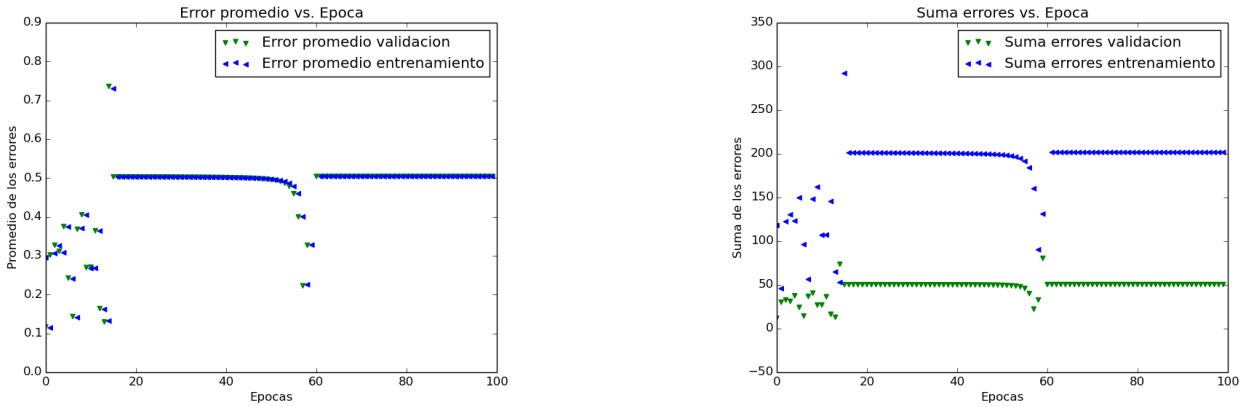

Figura 40: Errores en función de la época utilizando 2 nodos - Bipolar con método batch

Podemos ver que en las primeras épocas esta configuración presenta grandes fluctuaciones en el error, que por momentos se estanca (desde la época 20 hasta la 50) y luego continúa con un mismo valor, en particular desde la época 60 en adelante. En nuestras experimentaciones vimos que hasta la época 10000 esto no presentaba cambio alguno por lo cual creemos que realmente se estanca en ese valor. Al ser un error promedio cercano a 0.5, esta configuración termina siendo casi tan mala como las que veníamos observando en la primera parte de la experimentación.

Los detalles de cómo se vieron los errores por época se pueden observar en la siguiente tabla.

Epoca	1	10	50	100	1000	10000
Errorpromedio	0,2948447	0,4046325	0,4975718	0,5038568	0,5046087	0,5046087
Errorsuma	117,93789	161,85303	199,02875	201,54275	206,72043	206,72043

Cuadro 9: Tabla de errores en función de la época - Bipolar con método batch

Veamos ahora qué sucedió con 3 neuronas internas.

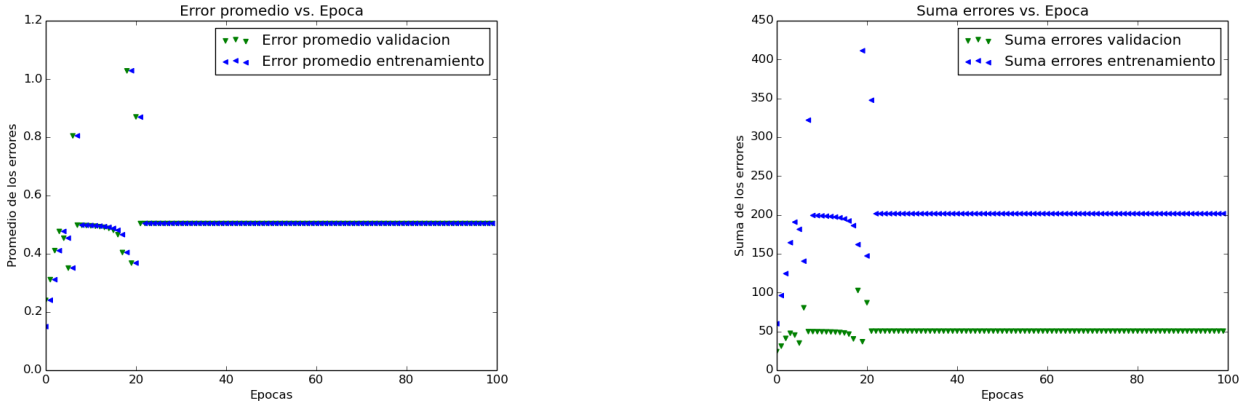

Figura 41: Errores en función de la época utilizando 3 nodos - Bipolar con método batch

Podemos observar la misma idea que en el gráfico anterior pero esta vez el primer período en el que

el error se estanca es más corto, durando desde la época 7 hasta la 17. Si se quiere, el beneficio de esta configuración es que se conoce el valor final en el que se estanca el error más tempranamente, hacia la época 25. De todas formas es un error del mismo orden que antes y por ende muy elevado.

Los detalles se pueden ver en la siguiente tabla.

Epoca	1	10	50	100	1000	10000
Errorpromedio	0,1499449	0,4976860	0,5039604	0,5039604	0,5039604	0,5039604
Errorsuma	59,977967	199,074422	201,584189	201,584189	201,584189	201,584189

Cuadro 10: Tabla de errores en función de la época - Bipolar con método batch

Por último presentamos los resultados de utilizar 4 neuronas intermedias.

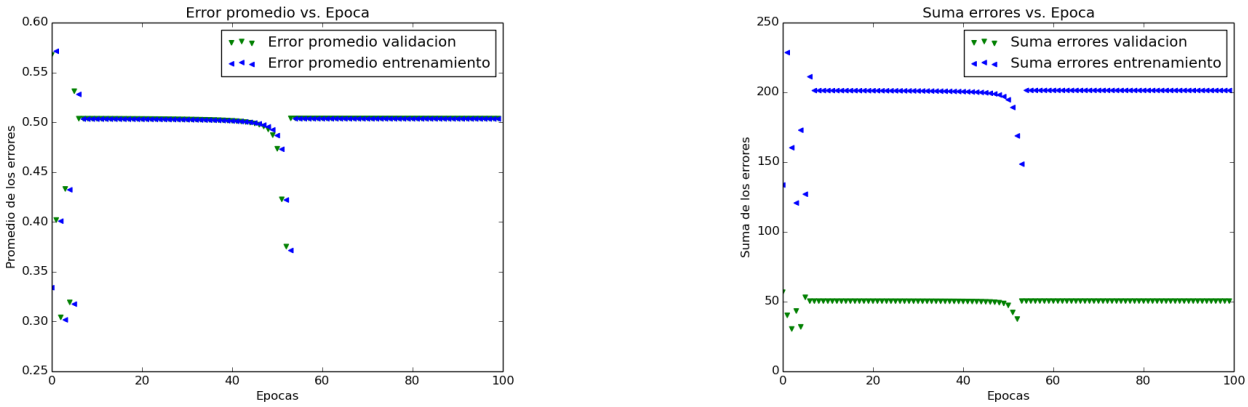

Figura 42: Errores en función de la época utilizando 4 nodos - Bipolar con método batch

Se puede observar un resultado similar al de la prueba con 2 neuronas intermedias, aunque esta vez la primera porción en la que se estanca comienza desde más temprana época. El error con el que creemos, en base a nuestra experimentación, converge, es del mismo orden que antes.

La última tabla que detalla esto se da a continuación.

Epoca	1	10	50	100	1000	10000
Errorpromedio	0,3339771	0,5031831	0,4923661	0,5033230	0,5045340	0,5049751
Errorsuma	133,59086	201,27325	196,94646	201,32922	202,234860	202,734122

Cuadro 11: Tabla de errores en función de la época - Bipolar con método batch

Finalmente concluímos que tomar pocos nodos no resulta en nada beneficioso para utilizar la sigmoidea *bipolar* con el método batch.

Cuarta etapa de experimentación

Veamos por último cómo resultó la cuarta prueba al entrenar una red con 30 nodos internos usando $\eta = 0{,}0001$ y función *bipolar*.

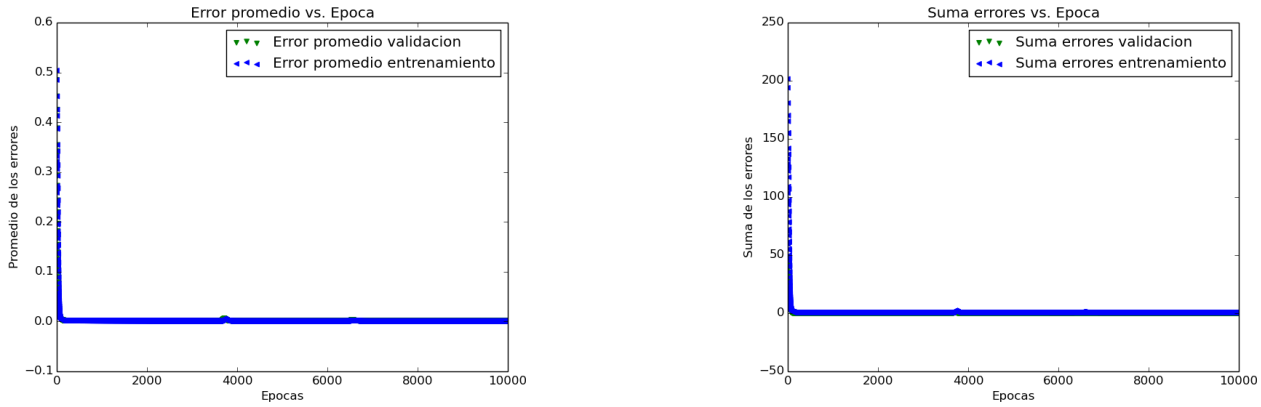

Figura 43: Errores en función de la época utilizando 30 nodos - Bipolar con método batch

Se puede observar que si bien los errores promedio comienzan siendo elevados (cercanos a 0.5) muy rápidamente se vuelven pequeños. Esto nos asombra en principio pues creíamos que combinar batch con la sigmoidea bipolar ocasionaba siempre errores altos.

Veamos a continuación una tabla que muestra los valores obtenidos con más detalle.

$Epoca$	1	10	50	100	1000	10000
$Errorpromedio$	0,5041414	0,3899460	0,0437125	0,0030932	0,0017842	0,0011171
$Errorsuma$	201,53468	168,34486	22,7075073	1,37188073	0,62984827	0,44686878

Cuadro 12: Tabla de errores en función de la época - Bipolar con método incremental

Con esto podemos ver que esta combinación de parámetros se comportó de forma muy similar a las combinaciones que destacamos en la primera etapa de experimentación. Sin embargo, se puede ver que tarda muchas épocas en estabilizarse a valores que con la otra configuración se conseguía con como mucho 5000.

4.2. Conclusiones

A raíz de toda la experimentación que realizamos, podemos concluir las siguientes ideas.

- Entrenar la red con este dataset utilizando la sigmoidea binomial genera muy buenas generalizaciones utilizando una capa interna de entre 5 y 15 neuronas. En particular, para conseguir una convergencia más rápida se puede optar con un valor de $\eta = 0{,}05$ y utilizar el método incremental con unas 2000 épocas máximas (o bien frenar hasta que el error sea del orden de 0.00115). También se puede utilizar el método batch con los mismos valores.
- No se debe entrenar la red utilizando la sigmoidea bipolar y el método batch con valores de $\eta \in \{0{,}05, 0{,}1, 0{,}2, 0{,}3, 0{,}4, 0{,}5\}$ y entre 5 y 15 neuronas en la capa intermedia dado que consigue errores muy elevados sin importar la cantidad de épocas que se lo deje correr.
- Si se quiere optar por utilizar la sigmoidea bipolar, creemos que sería conveniente utilizar valores $\eta \in \{0{,}01, 0{,}0001\}$ con 5 neuronas en la capa intermedia y una cantidad máxima de 20 épocas. Si se quiere utilizar mayor cantidad de neuronas en la capa intermedia, recomendamos utilizar 30 y un valor de $\eta = 0{,}0001$. De todas formas no pudimos profundizar mucho en estas configuraciones pero con la experimentación que realizamos ofreció muy buenos resultados, por lo cual basándonos en ésta diríamos que si se quiere ejecutar algo certero de forma muy rápida, se tendría que utilizar esta combinación de parámetros.

5. Ideas y Trabajo Futuro

Dataset 1: Cancer de mamas

Sobre cómo continuar el trabajo en este primer dataset:

- Para el modo incremental habría que probar con nuevos valores de η (más grandes y más chicos) para terminar de corroborar si existen tales que el entrenamiento funcione bastante más rápido. Para esto, aprovechar que ya tenemos un estimado en la cantidad de épocas para conseguir un error razonable, y hacer pruebas con esa cantidad de épocas y/o ese valor de error como criterios de parada. De esta forma, acelerar las corridas y la búsqueda de mejores configuraciones.
- Para el modo batch, sería interesante refinar la búsqueda para un valor de η que dé los mejores resultados posibles (probablemente partiendo de la base de que los valores con los que observamos los mejores resultados fueron 0.05 y 0.01, probar entonces con valores cercanos e intermedios).

Dataset 2: Eficiencia energética

Sobre cómo continuar el trabajo en el segundo dataset:

- Para el modo incremental creemos que habría que probar valores muy reducidos de η para así tener una mayor noción de los comportamientos del dataset, de la misma forma en la que lo hicimos en la segunda etapa de experimentación sobre batch.
- Para el modo batch, sería interesante hacer hincapié sobre la sigmoidea bipolar e intentar corroborar si efectivamente los resultados que obtuvimos al utilizar $\eta = 0{,}01$ y $\eta = 0{,}0001$ son similares tomando $\eta + \sigma$ para algún $\sigma > 0$ muy pequeño (es decir, si se comporta de la misma forma para parámetros muy cercanos al que utilizamos). De esta forma se podría saber fehacientemente que esta combinación de parámetros es la mejor, en base a nuestra experimentación.

Para la experimentación futura en general, el otro aspecto importante sería la arquitectura de la red, la cual para este trabajo dejamos fija en una única capa intermedia. Para el trabajo futuro, sería interesante analizar más variaciones de esto en una experimentación bastante más extensa, probando con varias capas intermedias y variando las cantidades de neuronas en cada una.

6. Secciones relevantes del código

6.1. main.cpp

```
1  import numpy as np
2  from scipy.special import expit
3  import math
4  import random
5  import string
6  import sys
7  import csv
8  import codecs
9  import os
10 from random import shuffle
11 import matplotlib.pyplot as plt
12 from time import time
13 #
    ------------------------------------------------------------------------------

14 def unicode_csv_reader(utf8_data, dialect=csv.excel, **kwargs):
15         csv_reader = csv.reader(utf8_data, dialect=dialect, **kwargs)
16         for row in csv_reader:
17             yield [unicode(cell, 'utf-8') for cell in row]
18         return
19 #
    ------------------------------------------------------------------------------

20 def destringifyTupleData(d):
21         return [tuple(destringifyList(l)) for l in d]
22 #
    ------------------------------------------------------------------------------

23 def destringifyList(l):
24         return map(float, l)
25 #
    ------------------------------------------------------------------------------

26 def sigmoid(x):
27         return 1.0/(1.0 + np.exp(-x))
28 #
    ------------------------------------------------------------------------------

29 def sigmoid_prima(x):
30         return sigmoid(x)*(1.0-sigmoid(x))
31 #
    ------------------------------------------------------------------------------

32 def tanh(x):
33         return np.tanh(x)
34 #
    ------------------------------------------------------------------------------

35 def tanh_prima(x):
```

```
36          return 1.0 - x**2
37  #
    ------------------------------------------------------------------------------


38  class redneuronal:
39          def __init__(self, capas,eta, funcionActivacion='tanh'):
40                  self.W = []
41                  self.deltaW = []
42                  self.LearningRate = eta
43                  self.capasCantidad = capas
44                  for i in range(1, len(capas) - 1):
45                          Wi = np.random.random((capas[i-1] + 1, capas[i] +
                                1))
46                          deltai = np.zeros((capas[i-1] + 1, capas[i] + 1))
47                          self.W.append(Wi)
48                          self.deltaW.append(deltai)
49                  Wn = np.random.random( (capas[len(capas)-2] + 1, capas[len(
                        capas) - 1] ))
50                  self.W.append(Wn)
51                  deltaN = np.zeros((capas[len(capas)-2] + 1, capas[len(capas
                        ) - 1] ))
52                  self.deltaW.append(deltaN)
53
54                  self.Ys = []
55                  for i in range(len(capas)):
56                          self.Ys.append([0]*capas[i])
57
58                  #Funcion de activacion
59                  self.funcion = funcionActivacion
60                  if funcionActivacion == 'sigmoid':
61                          self.factivacion = sigmoid
62                          self.factivacion_prima = sigmoid_prima
63                  elif funcionActivacion == 'tanh':
64                          self.factivacion = tanh
65                          self.factivacion_prima = tanh_prima
66  #
    ------------------------------------------------------------------------------


67          def correction(self,Z):
68                  error = Z - self.Ys[-1]
69
70                  echica = 0
71                  for i in range(len(error)):
72                          suma = error[i] * error[i]
73                          echica = echica  + suma
74
75                  #Calcular E
76                  E = [error * self.factivacion_prima(self.Ys[-1])]
77                  for l in range(len(self.Ys) - 2, 0, -1):
78                          E.append(E[-1].dot(self.W[l].T)  *   self.
                                factivacion_prima(self.Ys[l]))
79                  E.reverse() #asi iteramos mas facil
80
81                  #Calcular deltaW
```

```
82                 for i in range(len(self.W)):
83                         layer = np.atleast_2d(self.Ys[i])
84                         delta = np.atleast_2d(E[i])
85                         self.deltaW[i] = self.deltaW[i] + self.LearningRate
                               * layer.T.dot(delta)
86
87                 return echica
88 #
    ------------------------------------------------------------------------------

89         def activation(self,X):
90                 uno = np.ones(1).T
91                 X = np.concatenate((uno, X), axis=1)
92                 self.Ys[0] = X
93                 for i in range(len(self.W)):
94                         self.Ys[i+1] = self.factivacion(np.dot(self.Ys[i],
                            self.W[i]))
95                 return self.Ys[-1]
96 #
    ------------------------------------------------------------------------------

97         def adaptation(self):
98                 for j in range(len(self.W)):
99                         cantfilas = len(self.W[j])
100                        cantcolumnas = len(self.W[j][0])
101                        for fila in range(cantfilas):
102                                for columna in range(cantcolumnas):
103                                        self.W[j][fila][columna] = self.W[j
                                           ][fila][columna] + self.deltaW[j
                                           ][fila][columna]
104                                        self.deltaW[j][fila][columna] = 0;
105                return
106 #
    ------------------------------------------------------------------------------

107        def incremental(self, X,Z):
108                error = 0
109                p = len(X)
110                pos = [0]*p
111                pos = [i for i in range(p)]
112                shuffle(pos)
113                for i in range(p):
114                        self.activation(X[pos[i]])
115                        error = error + self.correction(Z[pos[i]])
116                        self.adaptation()
117                return error
118 #
    ------------------------------------------------------------------------------

119        def batch(self, X,Z):
120                error = 0
121                p = len(X)
122                pos = [0]*p
123                pos = [i for i in range(p)]
```

```
124                 shuffle(pos)
125                 for i in range(p):
126                         self.activation(X[pos[i]])
127                         error = error + self.correction(Z[pos[i]])
128                 self.adaptation()
129                 return error
130 #
    ------------------------------------------------------------------------------

131         def validacion_ninguna(self,X,Z,errorMin,maxEpocas,
                metodoEntrenamiento,directory,corrida):
132                 error=100
133                 epoca=0
134                 logrado = False
135
136                 directoryTiempos = directory + "/tiempos"+str(corrida) +".
                        txt"
137                 tiempos = open(directoryTiempos, 'w')
138
139                 directoryerrorEntrenamiento = directory + "/
                        errorEntrenamiento"+ str(corrida)+".txt"
140                 errorEntrenamiento = open(directoryerrorEntrenamiento, 'w')
141
142                 directoryerrorNEntrenamiento = directory + "/
                        errorNEntrenamiento"+ str(corrida)+".txt"
143                 errorEntrenamientoN = open(directoryerrorNEntrenamiento, 'w
                        ')
144
145                 start_time_global = time()
146
147                 while(error>errorMin and epoca<maxEpocas):
148                         start_time = time()
149                         if metodoEntrenamiento == 1 or metodoEntrenamiento
                                == 3:
150                                 error = self.incremental(X,Z)
151                         else: # vale 2 o 4
152                                 error = self.batch(X,Z)
153                         epoca =epoca +1
154                         print str(epoca) + " " + str(error)
155
156                         elapsed_time = time() - start_time
157                         tiempos.write(str(elapsed_time))
158                         tiempos.write("\n")
159                         print "Tiempo consumido: " + str(elapsed_time) + "
                                segundos."
160
161                         errorEntrenamiento.write(str(error))
162                         errorEntrenamiento.write("\n")
163
164                         error1 = error/ len(X)
165                         errorEntrenamientoN.write(str(error1))
166                         errorEntrenamientoN.write("\n")
167
168                 elapsed_time = time() - start_time_global
```

```
169                 tiempos.write(str(elapsed_time))
170                 tiempos.write("\n")
171                 tiempos.close()
172                 errorEntrenamiento.close()
173                 errorEntrenamientoN.close()
174
175                 directoryactivarEntrenamiento = directory + "/
                        activarEntrenamiento"+ str(corrida)+".txt"
176                 activarEntrenamiento = open(directoryactivarEntrenamiento,
                        'w') #append data
177
178                 for indice in range(len(X)):
179                                 Xout = X[indice]
180                                 Zout = Z[indice]
181                                 activarEntrenamiento.write(str(indice) + "
                                        resultado generado "+ str(self.
                                        activation(Xout)) + " resultado esperado
                                        " + str(Zout))
182                                 activarEntrenamiento.write("\n")
183
184                 print "Tiempo consumido global: " + str(elapsed_time) + "
                        segundos."
185
186                 if(error < errorMin and epoca < maxEpocas):
187                         logrado = True
188
189
190
191                 tam = epoca
192                 epocasTodas = np.arange(0, epoca,1)
193
194                 directoryTiempos = directory + "/tiempos"+str(corrida) +".
                        txt"
195                 tiempos = open(directoryTiempos, 'r') #read data
196
197                 directoryerrorEntrenamiento = directory + "/
                        errorEntrenamiento"+ str(corrida)+".txt"
198                 errorEntrenamiento = open(directoryerrorEntrenamiento, 'r')
                         #read data
199
200                 directoryerrorNEntrenamiento = directory + "/
                        errorNEntrenamiento"+ str(corrida)+".txt"
201                 errorEntrenamientoN = open(directoryerrorNEntrenamiento, 'r
                        ') #read data
202
203                 arrErrorEntrenamiento = [0]*tam
204                 arrTiempos = [0]*(tam)
205                 arrErrorEntrenamientoN = [0]*tam
206
207                 arrErrorEntrenamiento = arregloLeido(errorEntrenamiento,tam
                        )
208                 arrTiempos = arregloLeido(tiempos,tam)
209                 arrErrorEntrenamientoN = arregloLeido(errorEntrenamientoN,
                        tam)
```

```
210
211                 tiempos.close()
212                 errorEntrenamiento.close()
213                 errorEntrenamientoN.close()
214
215                 graficarSimple(epocasTodas, arrErrorEntrenamiento, "Suma
                        errores entrenamiento vs. Epoca", 'Suma de los errores de
                        entrenamiento', maxEpocas, "/
                        errorSumaEntrenamientoVsEpoca", directory, corrida)
216                 graficarSimple(epocasTodas, arrErrorEntrenamientoN, "Error
                        promedio entrenamiento vs. Epoca", 'Promedio de los
                        errores de entrenamiento', maxEpocas, "/
                        errorPromedioEntrenamientoVsEpoca", directory, corrida)
217                 graficarSimple(epocasTodas, arrTiempos, "Tiempos de las
                        epocas", 'Tiempos', maxEpocas, "/TiempoVsEpoca", directory,
                        corrida)
218
219                 ret = [0]*3
220                 ret[0] = logrado
221                 ret[1] = error
222                 ret[2] = epoca
223                 return ret
224 #
    ------------------------------------------------------------------------------

225         def testing(self,X, Z):
226                 error = 0
227                 for h in range(len(X)):
228                         Y = self.activation(X[h])
229                         agregado = 0
230                         for i in range(len(Y)):
231                                 agregado = agregado + (Z[h][i] - Y[i])*(Z[h
                                        ][i] - Y[i])
232                         error = error + agregado
233                 return error
234 #
    ------------------------------------------------------------------------------

235         def holdout(self,X,Z,errorMin,maxEpocas,r, metodoEntrenamiento,
                directory,corrida):
236                 p = len(X)
237                 pos = [0]*p
238                 pos = [i for i in range(p)]
239                 shuffle(pos)
240                 rp = (int)(r*p)
241                 resto = p - rp
242                 TX = [0]*rp
243                 TZ = [0]*rp
244                 VX = [0]*resto
245                 VZ = [0]*resto
246                 for i in range (rp):
247                         TX[i] = X[pos[i]]
248                         TZ[i] = Z[pos[i]]
249                 for i in range (resto):
```

```
250                     VX[i] = X[pos[i+rp]]
251                     VZ[i] = Z[pos[i+rp]]
252             error = 100
253             error_testing = 1
254             epoca = 0
255             logrado = False
256             directoryTiempos = directory + "/tiempos"+str(corrida) +".
                    txt"
257             tiempos = open(directoryTiempos, 'w')
258
259             directoryerrorEntrenamiento = directory + "/
                    errorEntrenamiento"+ str(corrida)+".txt"
260             errorEntrenamiento = open(directoryerrorEntrenamiento, 'w')
261
262             directoryerrorTesting = directory + "/errorTesting"+ str(
                    corrida)+".txt"
263             errorTesting = open(directoryerrorTesting, 'w')
264
265             directoryerrorNEntrenamiento = directory + "/
                    errorNEntrenamiento"+ str(corrida)+".txt"
266             errorEntrenamientoN = open(directoryerrorNEntrenamiento, 'w
                    ')
267
268             directoryerrorNTesting = directory + "/errorNTesting"+ str(
                    corrida)+".txt"
269             errorTestingN = open(directoryerrorNTesting, 'w')
270
271             start_time_global = time()
272             while(error > errorMin and epoca < maxEpocas):
273                     start_time = time()
274                     if metodoEntrenamiento == 1 or metodoEntrenamiento
                            == 3:
275                             error = self.incremental(TX,TZ)
276                     else: # vale 2 o 4
277                             error = self.batch(TX, TZ)
278                     error_testing = self.testing(VX, VZ)
279
280                     elapsed_time = time() - start_time
281                     tiempos.write(str(elapsed_time))
282                     tiempos.write("\n")
283                     print str(epoca) + "␣" + str(error)
284                     print "Tiempo␣consumido:␣" + str(elapsed_time) + "␣
                            segundos."
285                     errorEntrenamiento.write(str(error))
286                     errorEntrenamiento.write("\n")
287
288                     errorTesting.write(str(error_testing))
289                     errorTesting.write("\n")
290
291                     error1 = error/rp
292                     errorEntrenamientoN.write(str(error1))
293                     errorEntrenamientoN.write("\n")
294
295                     error_testing1 = error_testing/resto
```

```
296                         errorTestingN.write(str(error_testing1))
297                         errorTestingN.write("\n")
298
299                         epoca = epoca +1
300                 elapsed_time = time() - start_time_global
301                 tiempos.write(str(elapsed_time))
302                 tiempos.write("\n")
303                 tiempos.close()
304                 errorEntrenamiento.close()
305                 errorTesting.close()
306                 errorEntrenamientoN.close()
307                 errorTestingN.close()
308
309
310                 directoryactivarEntrenamiento = directory + "/
                        activarEntrenamiento"+ str(corrida)+".txt"
311                 activarEntrenamiento = open(directoryactivarEntrenamiento,
                        'a') #append data
312
313                 directoryactivarTesting = directory + "/activarTesting"+
                        str(corrida)+".txt"
314                 activarTesting= open(directoryactivarTesting, 'a') #append
                        data
315
316                 for indice in range(p):
317                         Xout = X[pos[indice]]
318                         Zout = Z[pos[indice]]
319                         if (indice < rp):
320                                 activarEntrenamiento.write(str(pos[indice])
                                     + " resultado generado "+ str(self.
                                    activation(Xout)) + " resultado esperado
                                    " + str(Zout))
321                                 activarEntrenamiento.write("\n")
322 #                               activarEntrenamiento.write(str(self.
    activation(Xout)[0]) + " " + str(Zout[0]))
323 #                               activarEntrenamiento.write("\n")
324
325                         else:
326                                 activarTesting.write(str(pos[indice]) + "
                                    resultado generado "+ str(self.
                                    activation(Xout)) + " resultado esperado
                                    " + str(Zout))
327                                 activarTesting.write("\n")
328 #                               activarTesting.write(str(self.activation(
    Xout)) + " " + str(Zout))
329 #                               activarTesting.write("\n")
330
331                 if(error < errorMin and epoca < maxEpocas):
332                         logrado = True
333
334                 tam = epoca
335                 epocasTodas = np.arange(0, epoca,1)
336
337                 directoryTiempos = directory + "/tiempos"+str(corrida) +".
```

```
                txt"
338             tiempos = open(directoryTiempos, 'r') #read data
339
340             directoryerrorEntrenamiento = directory + "/
                    errorEntrenamiento"+ str(corrida)+".txt"
341             errorEntrenamiento = open(directoryerrorEntrenamiento, 'r')
                     #read data
342
343             directoryerrorTesting = directory + "/errorTesting"+ str(
                    corrida)+".txt"
344             errorTesting = open(directoryerrorTesting, 'r') #read data
345
346             directoryerrorNEntrenamiento = directory + "/
                    errorNEntrenamiento"+ str(corrida)+".txt"
347             errorEntrenamientoN = open(directoryerrorNEntrenamiento, 'r
                    ') #read data
348
349             directoryerrorNTesting = directory + "/errorNTesting"+ str(
                    corrida)+".txt"
350             errorTestingN = open(directoryerrorNTesting, 'r') #read
                    data
351
352             arrErrorTesting = [0]*tam
353             arrErrorEntrenamiento = [0]*tam
354             arrTiempos = [0]*(tam)
355             arrErrorEntrenamientoN = [0]*tam
356             arrErrorTestingN = [0]*tam
357
358             arrErrorTesting = arregloLeido(errorTesting,tam)
359             arrErrorEntrenamiento = arregloLeido(errorEntrenamiento,tam
                    )
360             arrTiempos = arregloLeido(tiempos,tam)
361             arrErrorEntrenamientoN = arregloLeido(errorEntrenamientoN,
                    tam)
362             arrErrorTestingN = arregloLeido(errorTestingN,tam)
363
364             tiempos.close()
365             errorEntrenamiento.close()
366             errorTesting.close()
367             errorEntrenamientoN.close()
368             errorTestingN.close()
369
370             graficarDoble(epocasTodas,arrErrorTesting,
                    arrErrorEntrenamiento,'Suma␣errores␣validacion','Suma␣
                    errores␣entrenamiento',"Suma␣errores␣vs.␣Epoca",'Suma␣de
                    ␣los␣errores',maxEpocas,"/errorSumaVsEpoca",directory,
                    corrida)
371             graficarSimple(epocasTodas,arrErrorEntrenamiento,"Suma␣
                    errores␣entrenamiento␣vs.␣Epoca",'Suma␣de␣los␣errores␣de
                    ␣entrenamiento',maxEpocas,"/
                    errorSumaEntrenamientoVsEpoca",directory,corrida)
372             graficarSimple(epocasTodas,arrErrorTesting,"Suma␣errores␣
                    validacion␣vs.␣Epoca",'Suma␣de␣los␣errores␣de␣validacion
                    ',maxEpocas,"/errorSumaTestingVsEpoca",directory,corrida
```

```
                    )
373                 graficarDoble(epocasTodas, arrErrorTestingN,
                        arrErrorEntrenamientoN, 'Error promedio validacion', '
                        Error promedio entrenamiento',"Error promedio vs. Epoca"
                        ,'Promedio de los errores',maxEpocas,"/
                        errorPromedioVsEpoca",directory,corrida)
374                 graficarSimple(epocasTodas, arrErrorEntrenamientoN,"Error
                        promedio entrenamiento vs. Epoca",'Promedio de los
                        errores de entrenamiento',maxEpocas,"/
                        errorPromedioEntrenamientoVsEpoca",directory,corrida)
375                 graficarSimple(epocasTodas, arrErrorTestingN,"Error promedio
                        validacion vs. Epoca",'Promedio de los errores de
                        validacion',maxEpocas,"/errorPromedioTestingVsEpoca",
                        directory,corrida)
376                 graficarSimple(epocasTodas, arrTiempos,"Tiempos de las
                        epocas",'Tiempos',maxEpocas,"/TiempoVsEpoca",directory,
                        corrida)
377
378                 print "Tiempo consumido global: " + str(elapsed_time) + "
                        segundos."
379                 ret = [0]*4
380                 ret[0] = logrado
381                 ret[1] = error
382                 ret[2] = epoca
383                 ret[3] = error_testing
384                 return ret
385 #
```

```
386 def ej1(sigmoidea):
387         if(sigmoidea == 1):
388                 filename = '../tp1_training_dataset_1m.csv'
389                 print "sigmoidea"
390         elif(sigmoidea == 2):
391                 filename = '../tp1_training_dataset_1mb.csv'
392                 print "bipolar"
393         x = []
394         z = []
395         reader = unicode_csv_reader(open(filename))
396         fileencoding = "iso-8859-1"
397         primero = True
398         for resultado,field1, field2, field3, field4, field5, field6,
              field7, field8, field9, field10, field11, field12, field13,
              field14, field15, field16, field17, field18, field19, field20,
              field21, field22, field23, field24, field25, field26, field27,
              field28, field29, field30 in reader:
399                 if (primero):
400                         primero = False
401                 else:
402                         raw = [(float)(field1),(float)(field2),(float)(
                                field3),(float)(field4),(float)(field5),(float)(
                                field6),(float)(field7),(float)(field8),(float)(
                                field9),(float)(field10),(float)(field11),(float
                                )(field12),(float)(field13),(float)(field14),(
```

```
                                float)(field15),(float)(field16),(float)(field17
                                ),(float)(field18),(float)(field19),(float)(
                                field20),(float)(field21),(float)(field22),(
                                float)(field23),(float)(field24),(float)(field25
                                ),(float)(field26),(float)(field27),(float)(
                                field28),(float)(field29),(float)(field30)]
403                         norm = [float(i)/sum(raw) for i in raw]
404                         x.append(norm)
405                         z.append([(float)(resultado)])
406         x = destringifyTupleData(x)
407         ret = [0]*2
408         ret[0] = x
409         ret[1] = z
410         return ret
411 #
    ------------------------------------------------------------------------------

412 def arregloLeido(fileRead,tam):
413         arreglo = [0]*(tam)
414         i = 0
415         for line in fileRead:
416                 if(i != tam):
417                         arreglo[i] = (float)(line)
418                 i = i + 1
419         return arreglo
420 #
    ------------------------------------------------------------------------------

421 def graficarDoble(epocasTodas,arrErrorTesting,arrErrorEntrenamiento,lbl1,
        lbl2,titulo,ylabel,maxEpocas,guardar,directory,corrida):
422         colors=['b','g','y','c', 'k', 'm', 'r','chartreuse','burlywood','#
                ffbbee']
423         markers = ['^','v','<','>','x','+','o','p','h','s']
424         plt.subplot()
425         plt.scatter(epocasTodas, arrErrorTesting,label=lbl1, marker =
                markers[1], color=colors[1])
426         plt.scatter(epocasTodas, arrErrorEntrenamiento,label=lbl2, marker =
                 markers[2], color=colors[0])
427         plt.title(titulo)
428         plt.legend(loc='upper right')
429         plt.xlabel('Epocas')
430         plt.ylabel(ylabel)
431         plt.xlim(0, maxEpocas)
432         directoryerrorEntrenamiento = directory + guardar+ str(corrida)+".
                png"
433         plt.savefig(directoryerrorEntrenamiento)
434 # plt.show()
435         plt.close()
436         return
437 #
    ------------------------------------------------------------------------------

438 def graficarSimple(epocasTodas,arrErrorEntrenamiento,titulo,ylabel,
        maxEpocas,guardar,directory,corrida):
```

```
439         colors=['b','g','y','c', 'k', 'm', 'r','chartreuse','burlywood','#
                ffbbee']
440         markers = ['^','v','<','>','x','+','o','p','h','s']
441         plt.subplot()
442         plt.scatter(epocasTodas, arrErrorEntrenamiento, marker = markers
                [6], color=colors[0])
443         plt.title(titulo)
444         plt.xlabel('Epocas')
445         plt.ylabel(ylabel)
446         plt.xlim(0, maxEpocas)
447         directoryerrorEntrenamiento = directory + guardar+ str(corrida)+".
                png"
448         plt.savefig(directoryerrorEntrenamiento)
449         #plt.show()
450         plt.close()
451         return
452 #
    ------------------------------------------------------------------------------

453 def ej2():
454         filename = '../tp1_training_dataset_2m.csv'
455         x = []
456         z = []
457         reader = unicode_csv_reader(open(filename))
458         fileencoding = "iso-8859-1"
459         primero = True
460         for data1,data2,data3,data4,data5,data6,data7,data8,res1,res2 in
                reader:
461                 if (primero):
462                         primero = False
463                 else:
464                         raw = [(float) (data1),(float)(data2),(float)(data3
                                ),(float)(data4),(float)(data5),(float)(data6),(
                                float)(data7),(float)(data8)]
465                         norm = [float(i)/sum(raw) for i in raw]
466                         x.append(norm)
467                         raw2 = [(float)(res1),(float)(res2)]
468                         norm2 = [float(i)/sum(raw2) for i in raw2]
469                         z.append(norm2)
470         x = destringifyTupleData(x)
471         z = destringifyTupleData(z)
472         ret = [0]*2
473         ret[0] = x
474         ret[1] = z
475         return ret
476 #
    ------------------------------------------------------------------------------

477 def test(red,X):
478         for e in X:
479                 print(e,red.activation(e))
480 #
    ------------------------------------------------------------------------------
```

```
481 def main(argv):
482         #sys.argv[1] Base de datos                                    1 o
                2
483         #sys.argv[2] Eta
                        entre 0 y 1
484
485         #sys.argv[3] errorMinimo
                numerito
486         #sys.argv[4] epocasEntrenamietno                    numerito
487         #sys.argv[5] modo   1.hold out incremental 2.hold out batch 3.
                validacion ninguna incremental 4.validacion ninguna batch
488         #sys.argv[6] cuanto le doy a entrenamiento? num entre 0 y 1
489         #sys.argv[7] funcion sigmoidea 1.sigmoidea 2.tanh
490         #sys.argv[8] corridas
491
492         #sys.argv[9] cantidad de capas internas
493         #sys.argv[10...] cantidad de nodos en capas internas
494
495         bd = int(sys.argv[1])
496         eta = float(sys.argv[2])
497         errorMinimo = float(sys.argv[3])
498         epocasEntrenamietno = int(sys.argv[4])
499         modo = int(sys.argv[5])
500         r = float(sys.argv[6])
501         sigmoidea = int(sys.argv[7])
502         corridas = int(sys.argv[8])
503         cantidad = int(sys.argv[9])
504         capas = [0]*(2+cantidad)
505         if(cantidad >0):
506                 nodos = [0]*cantidad
507                 for i in range(cantidad):
508                         nodos[i] = int(sys.argv[10+i])
509                         capas[i+1] = nodos[i]
510         else:
511                 nodos = []
512         if(bd == 1):
513                 ret = ej1(sigmoidea)
514                 capas[0] = 30
515                 capas[-1] = 1
516         elif(bd == 2):
517                 ret = ej2()
518                 capas[0] = 8
519                 capas[-1] = 2
520         elif(bd == 3):
521                 capas[0] = 2
522                 capas[-1] = 1
523         if(bd == 1 or bd == 2):
524                 x = ret[0]
525                 z = ret[1]
526         else:
527                 x = np.array([[0, 0],[0, 1],[1, 0],[1, 1]])
528                 z = np.array([0, 1, 1, 0])
529         if(sigmoidea == 1):
530                 directory = "TP1.BD"+str(bd)+".Eta"+str(eta)+".errorMinimo"
```

```
                    +str(errorMinimo)+".epocasEntrenamietno"+str(
                    epocasEntrenamietno)+".modo"+str(modo)+".Entrenamiento"+
                    str(r)+".sigmoid.capasInternas"+str(cantidad)
531         else:
532                 directory = "TP1.BD"+str(bd)+".Eta"+str(eta)+".errorMinimo"
                    +str(errorMinimo)+".epocasEntrenamietno"+str(
                    epocasEntrenamietno)+".modo"+str(modo)+".Entrenamiento"+
                    str(r)+".tanh.capasInternas"+str(cantidad)
533         if(cantidad >0):
534                 for i in range(cantidad):
535                         aux = str(nodos[i])
536                         directory = directory +"."+ aux
537         if not os.path.exists(directory):
538                 os.makedirs(directory)
539         for i in range(corridas):
540                 if(sigmoidea == 1):
541                         red = redneuronal(capas,eta,"sigmoid")
542                 else:
543                         red = redneuronal(capas,eta,"tanh")
544                 if(modo == 1):
545                         ret = red.holdout(x,z,errorMinimo,
                            epocasEntrenamietno,r,1,directory,i)
546                 elif(modo == 2):
547                         ret = red.holdout(x,z,errorMinimo,
                            epocasEntrenamietno,r,2,directory,i)
548                 elif(modo == 3):
549                         ret = red.validacion_ninguna(x,z,errorMinimo,
                            epocasEntrenamietno, 1,directory,i)
550                 elif(modo == 4):
551                         ret = red.validacion_ninguna(x,z,errorMinimo,
                            epocasEntrenamietno, 2,directory,i)
552                 if(modo == 3 or modo == 4):
553                         if(ret[0]):
554                                 print "Se logro con una suma de errores de
                                    " + str(ret[1]) + " en " + str(ret[2]) +
                                     "epocas!"
555                         else:
556                                 print "No se pudo entrenar la red y se
                                    obtuvo una suma de error de" + str(ret
                                    [1]) + "."
557                 elif(modo == 1 or modo == 1):
558                         if(ret[0]):
559                                 print "Se logro con una suma error de
                                    entrenamiento de" + str(ret[1]) + " y
                                    una suma de error de testing de"+ str(
                                    ret[3]) + " en " + str(ret[2]) + "
                                    epocas!"
560                         else:
561                                 print "No se pudo entrenar la red y se
                                    obtuvo una suma de error de
                                    entrenamiento de" + str(ret[1]) + " y
                                    una suma de error de testing de" + str(
                                    ret[3])
562                 if(bd == 3):
```

```
563                             test(red,x)
564 if __name__ == '__main__':
565         main(sys.argv)
```