Instituto Tecnológico de Buenos Aires

Sistemas de Inteligencia Artificial

Ingeniería en Informática

Redes Neuronales Multicapa

Primer Trabajo Práctico Especial

Titular: Parpaglione, Cristina

Semestre: 2018A

Grupo: 4

Repositorio: https://bitbucket.org/itba/sia-2018-04

Fecha: 26 de marzo de 2018

Entrega: 26 de marzo de 2018 a las 10:00hs

Autores: Delgado, Francisco (#57101)

Buscaglia, Matías Alejandro (#53551)

Gonzalez, Lautaro Nicolás (#54315)

Costesich, Pablo Alejandro (#50109)

Resumen

Se presenta un trabajo práctico de Redes Neuronales Multicapas para la detección y aprendizaje de datos topográficos. Se realiza la implementación de la red en el software de cálculo Octave, se analiza el funcionamiento de la misma y se extraen conclusiones en base a la arquitectura de la red obtenida de las diversas iteraciones realizadas durante la etapa de investigación. Se investiga la aplicación de diversas técnicas de optimización como *Momentum* y *Adaptive* η, así como también distintas distribuciones de neuronas y capas.

Índice

Índice de cuadros

Índice de figuras

1. Enunciado

Una empresa de videojuegos precisa un desarrollo que pueda simular en su plataforma terrenos de diferentes partes del mundo a partir de mediciones de altura, latitud y longitud que el equipo de simulación ha tomado de los terrenos reales. La red neuronal multicapa que se implemente deberá poder aproximar lo mejor posible el terreno, de manera que este parezca real. Cada grupo debe construir los patrones de entrenamiento a partir del muestreo obtenido.

1.1. Archivos de soporte

En el campus se encuentra el archivo `terrain.zip`, el mismo contiene los archivos de ejemplo con los puntos antes mencionados. El archivo que le corresponde a cada grupo es `terrainN.data`, donde N es el número del grupo.

1.2. Presentación

En la presentación se les puede dar un conjunto nuevo de puntos para ver como generaliza la red implementada.

1.3. Informe

El informe deberá analizar los resultados obtenidos con las distintas configuraciones, mostrar y justificar cuál es la configuración óptima para el problema dado. No está permitido el uso de librerías o toolkits de redes neuronales. Cualquier recurso tercero utilizado deberá ser consultado con la cátedra previo a su inclusión.

1.4. Puntos a desarrollar

Para el desarrollo, deben considerarse los siguientes puntos:

- Realizar un archivo de configuración, de forma tal que los aspectos relevantes de la red neuronal sean parametrizables.
- Utilizar como funciones de activación tanto la exponencial como la tangente hiperbólica y compararlas.
- Utilizar distintas arquitecturas de red y compararlas. Decir justificando en el informe con qué arquitectura aprendió mejor.
- Graficar la salida de la red neuronal de forma tal de analizar la función en el dominio que abarcan los puntos de entrada.
- Graficar el error (entrenamiento y testeo) de la red neuronal en su evolución.
- Obtener cualquier otra métrica que sea de utilidad para la defensa del funcionamiento de la red neuronal.
- Analizar la capacidad de generalización de la red.
- De las mejoras al algoritmo backpropagation implementar al menos *Momentum* y *Eta adaptativo*. Explicar cuáles fueron las elegidas. Comparar los resultados obtenidos con y sin estas mejoras.
- Considerar otras mejoras de optimización y precisión discutidas en clase.

2. Introducción

Se utiliza una Red Neuronal Multicapa de aprendizaje supervisado, con una única neurona de salida y dos de entrada. La misma tiene un número arbitrario de capas y de neuronas ocultas, que son determinados para el problema en base a parámetros obtenidos mediante experimentación.

El entrenamiento de la red se logra en base a un subconjunto de los casos aportados en la información topográfica relevada por la empresa. Aquellos datos no utilizados en el entrenamiento son empleados para calcular el nivel de generalización. La proporción es elegida de forma tal que permite un buen nivel de aprendizaje y una cantidad estadísticamente significativa de datos de prueba.

Para la determinación de la red se define la configuración inicial: si se utiliza *Adaptive* η o *Momentum* (o ambas), la función de activación g empleada, si el entrenamiento es en *batch* o *incremental* y la arquitectura según neuronas por capa. En base a estos parámetros se obtiene una lista de matrices que representan los pesos de las conexiones entre capas.

Luego, se utiliza el algoritmo de *back-propagation* para *Multi-Layer Perceptron Network* sobre los datos del mapa. El proceso es, de manera simplificada, el siguiente:

- Por época, se toma un subconjunto de los datos para el entrenamiento y se aplican a la red.
- Para cada patrón se realiza una propagación hacia adelante (*forward-propagation*) y se obtiene el resultado. La diferencia entre éste y el esperado se almacena como error.
- El error es propagado hacia atrás (*back-propagation*), que resulta en un vector de ajustes para cada capa. Aquí existe una diferencia sutil entre *batch* e *incremental*: en el primero se suma el *delta weight* de cada capa y se actualiza al terminar el batch, mientras que en el segundo se actualizan los pesos de las capas entre patrones.

- Al finalizar una época, se calcula el error de generalización y el error de aprendizaje. El mismo se grafica a fines de ilustrar posibles mesetas en el aprendizaje.
- Se actualiza el valor del η y el *momentum*, si es que dicha funcionalidad se encuentra activa.

El resultado de cada época es un nuevo vector de matrices de pesos, y determina una red neuronal entrenada. Se parte de una red anterior para entrenar una nueva. Se debe finalizar luego de lograr un error apreciablemente pequeño, o superada una cantidad arbitraria de generaciones.

2.1. Modelo de la Red Neuronal Multicapa

La red neuronal multicapa se determina en base a una arquitectura y su vector de matrices de pesos. El proceso para entrenarla requiere la determinación del modo de actualización de los pesos (*batch* o *incremental*) y de la función de activación.

Se establece la relación entre las neuronas en base a una matriz que indica los pesos de cada conexión. Por lo tanto, definimos la arquitectura como un vector $A \in \mathbb{N}^n$, que representa una red de n capas. Es importante notar que la arquitectura no contempla las neuronas *bias*.

Se define el conjunto de sinapsis entre dos capas como una matriz $W_c = \mathbb{R}^{m \times n}$, donde m es la cantidad de neuronas en la capa inferior (entrada, c) y n la superior (de salida, $c+1$). Se debe considerar que la capa de entrada contiene $m = x + 1$ conexiones, con x número de neuronas especificado en la arquitectura para la capa de entrada ya que agrega la neurona *bias*.

2.2. Determinación de Potencial de Activación

Se utiliza para la ecualización de los potenciales de activación funciones continuas y diferenciables. Se seleccionan dos funciones de activación basadas en que cuya derivada se define en función de su valor, lo que resulta de gran practicidad a la hora de su implementación

2.2.1. Sigmóidea Exponencial y su Derivada

Sea $SigExp : \mathbb{R} \to [0, 1]$:

$$SigExp(x) = (1 + e^{-x})^{-1}$$

Se define para $\vec{v}$ vector de n elementos:

$$\vec{SigExp}(\vec{v}) = \{SigExp(v_1), \cdots, SigExp(v_n)\}$$

Queda, entonces, definida su derivada por:

$$SigExp'(x) = x(1 - x)$$

Y su derivada como la aplicación componente a componente al vector.

2.2.2. Tangente Hiperbólica y su Derivada

De manera similar, sea $TanH : \mathbb{R} \to [-1, 1]$:

$$TanH(x) = \frac{\sinh x}{\cosh X} = \tanh x$$

Su aplicación a vectores es, al igual que en $SigExp$, componente a componente.

Su derivada queda definida como:

$$TanH'(x) = 1 - x^2$$

Su aplicación a vectores es componente a componente.

2.3. Aplicación de un patrón a la Red Neuronal Multicapa

La aplicación de un patrón (denominado *forward*) resulta de un producto de transformaciones sobre matrices de sinapsis, del que se obtiene un vector como resultado.

Sea $\xi^m_{entrada}$ el vector de entrada para una capa m. Se define $\xi^m = \xi^m_{entrada} : -1$ como la concatenación de $\xi^m_{entrada}$ con -1. Esto resulta de la adición de la neurona *bias*, que resulta implícita en la arquitectura.

Sea el potencial de membrana definido como la aplicación de un vector de entrada a una capa:

$$h^n = \xi^m \cdot W_n$$

El resultado es la aplicación de los impulsos a las neuronas, resultando en salidas sin ser procesadas por la función de activación g. Por lo tanto, es necesario aplicar g.

Sea el vector efectivo de resultados (salidas calculadas):

$$O^{capa} = g(h^{capa})$$

En general, la aplicación de un paso *forward* resulta en:

$$g\left(g\left(g\left(\cdots\right) \cdot W_{N-2}\right) \cdot W_{N-1}\right) = O$$

Con el caso base siendo la excepción a la regla, sobre la que no se le aplica la función de potencial de activación previo multiplicarla:

$$g\left(\xi^1 \cdot W_1\right)$$

2.4. Error para un patrón según Aprendizaje Supervisado

El cálculo del error resulta de realizar la resta de los valores obtenidos por los esperados. Este es un error por componente de salida, y se calcula como $\delta^{out} = g'(h^N) \cdot (S - O)$, con N índice de la capa de salida.

2.5. Propagación del error por *back-propagation*

Se define el algoritmo de *back-propagation* como un paso de actualización de las matrices de sinapsis.

2.5.1. Determinación de Deltas

Se define el vector de deltas δ^n para una matriz de sinapsis transpuesta n como:

$$\delta^n = g'(h^{n+1}) \odot (W'_n \cdot \delta^{n+1})$$

Con $\odot$ producto elemento a elemento o Producto Hadamard (Million, 2007).

Existe un vector de deltas correspondiente para cada capa de la red neuronal. Es importante notar que en la última capa el ajuste resulta de utilizar $\delta^{n+1} = \delta^{out}$.

2.5.2. *Adaptive* η

Se define η como un parámetro de aprendizaje, similar a la plasticidad neuronal en un cerebro biológico. El mismo define el proporción con el que se aplica la corrección sobre los pesos de la matriz sináptica para cada capa.

Adaptive η es una modificación al parámetro de aprendizaje tal que éste depende del error de cada época. Se define la variación de η de la siguiente forma:

- $\Delta E < 0$: $\eta = \eta\alpha$
- $\Delta E > \epsilon$: $\eta = \eta\beta$
- $\epsilon > \Delta E > 0$: $\eta = \eta$

Es decir, si el error decrece, el η aumenta en un porcentaje definido por α. Si el error aumenta muy poco, entonces el mismo no es alterado, pero si pasa un umbral definido por ϵ entonces se reduce la plasticidad.

2.5.3. *Momentum*

Momentum es el análogo de aplicar inercia sobre una caminata de un terreno. Se calcula para la capa n como:

$$Momentum^n = \alpha \cdot \Delta W_{n-1}$$

Es importante notar que el valor de α se configura a cero una vez que el error supera el umbral de ϵ, y a su valor original cuando el error se reduce.

2.5.4. Determinación de Nuevos Pesos

Las variaciones sobre los pesos se dan como el producto de la matriz de pesos anterior (W'_n) y el vector δ^n.

$$\Delta W_n = W'_n \cdot \delta^n$$

Los nuevos pesos de la capa para la red entrenada ($W^\star_n$) se definen como la suma de los pesos de la matriz sináptica con los aportes proporcionales: $W^\star_n = W_n + \Delta W_n$.

2.6. Evaluación de una Época

Por época, se toma un subconjunto de los datos para el entrenamiento y se aplican a la red. El resto de los patrones se considera un subconjunto de pruebas, que es utilizado para la determinación de los errores de aprendizaje y de generalización.

Por cada patrón de entrenamiento, se realiza un paso *forward* y un paso *backward*. La actualización de los pesos de la red (las matrices sinápticas) es un detalle propio del método seleccionado: *batch* o *incremental*.

Al obtener la nueva red, se toman todos los valores de entrenamiento y se calcula el error cuadrático medio. Ello determina el error de aprendizaje o aproximación de la red: la suma de todos los errores individuales determina el error de aprendizaje.

Asimismo, con la nueva red también se calcula el error de generalización: se toman los patrones que no fueron utilizados para entrenar la red (de prueba) y se obtiene el error cuadrático medio.

Se actualiza el valor del η y el *momentum*, si es que dicha funcionalidad se encuentra activa, utilizando el error de aprendizaje.

2.6.1. Proceso *incremental*

El proceso incremental mejora iterativamente con cada patrón de entrenamiento la red obtenida, que a su vez es utilizada como parámetro de los siguientes patrones. Esto permite un buen ajuste según los datos, pero como desventaja principal se cuenta que no resulta paralelizable.

2.6.2. Proceso *batch*

El proceso *batch* utiliza siempre la misma configuración de matrices sinápticas para todos los patrones de entrenamiento. Los ΔW obtenidos por (capa, patrón) son sumados elemento a elemento (agrupándolos por capa) a modo de conseguir un vector de matrices de ajustes. La nueva red neuronal consta de las matrices sinápticas ajustadas por los diferenciales.

Este proceso tiene como ventaja ser altamente paralelizable, pero su aplicación puede resultar en casos donde múltiples patrones anulen los aportes de ajustes entre sí, perdiendo información importante para disminuir el error final de aprendizaje.

3. Implementación

3.1. Archivo de configuración

Para el trabajo práctico se define el archivo de configuración *main.m* que contiene variables globales las cuales se pueden modificar para modificar el comportamiento del programa.

Se permite definir variables esenciales para el entrenamiento de la red tal como el factor de aprendizaje η, la función de activación a utilizar, el porcentaje de casos que se utilizarán para entrenamiento teniendo en cuenta que el complemento de este conjunto se usará para testeo de generalización, si se utiliza aprendizaje en batch o incremental y la arquitectura de la red.

Además de estos parámetros esenciales se permite activar y modificar el comportamiento de las mejoras aplicadas al algoritmo de entrenamiento, estas mejoras siendo η adaptativo y momentum. En cuanto a momentum se permite definir si este se utilizará, y en el caso de ser así también se permite definir el α que utilizará momentum. En el caso del eta adaptativo se puede modificar los valores de a y b que utiliza, la cantidad k de pasos con error decreciente para que se empiece a incrementar el η, y el porcentaje de error por el que tiene que superar el nuevo error al anterior para que decrezca el η.

3.2. Funciones de procesamiento de datos

Se utiliza la función `dlmread(input-file-name)` para la carga de datos. La misma es capaz de cargar el archivo de data directamente.

Se descarta una fila ya que son los identificadores (*header*). No se realiza preprocesamiento de los datos de entrada posteriormente.

3.3. Funciones de activación

Se definen dos funciones con sus derivadas, basadas en las clases teóricas: sigmóidea exponencial y tangente hiperbólica. Su uso, tal como fue explicitado en la introducción, se debe a que el valor de su derivada guarda relación con su valor propio.

3.3.1. Sigmóidea Exponencial

Se define la función como:

```
function ret = sig_exp(z)
  ret = 1 ./ (1 + e.^-z);
endfunction
```

Que usa la propiedad de actuar elemento a elemento en un vector. Su derivada es definida *in-line* en la función g en base al valor propio.

3.3.2. Tangente Hiperbólica

Se re-utiliza la definida por Octave, ya que la misma actúa sobre vectores de forma elemento a elemento. Su derivada es definida *in-line* en la función g en base al valor propio.

3.4. Funciones de instanciación de la red

La función `init` se encarga de la instanciación de la red. La misma retorna un vector de matrices sinápticas, inicializadas con pesos uniformemente distribuidos entre $\left[-\sqrt{\frac{6}{n+m+1}};\sqrt{\frac{6}{n+m+1}}\right]$, con n y m siendo las neuronas en el vector de arquitecturas de capas consecutivas. Se agrega una extra por la neurona *bias*.

3.5. Función de entrenamiento

El archivo `train.m` contiene la función homónima, que realiza tanto el proceso *batch* como el *incremental*.

3.5.1. Paso *Forward*

Es una función que se encuentra separada en el caso base (sólo se tiene el vector de matrices sinápticas y la entrada), y los casos internos que constan de la propagación entre capas de los valores.

La misma se encarga de normalizar los vectores de entrada al largo deseado, agregando la neurona de *bias*, y de retornar el vector de resultados sin el *bias*.

3.5.2. Paso *Back*

El paso back se encuentra, al igual que *Forward*, separado entre el caso base donde recibe el patrón deseado y el obtenido, de los casos internos de propagación de errores.

El resultado de la función es un nuevo vector con matrices sinápticas actualizadas, así como los diferenciales que se aplicaron sobre las mismas. Esto permite realizar un esquema iterativo *incremental* con sólo modificar el momento de actualización de las matrices sinápticas sin tener que escribir dos funciones separadas por estrategia.

4. Resultados

En la siguiente seccion buscamos contrastar y comparar como la variacion de ciertos aspectos que definen a la red afectan su capacidad de memorizar y generalizar los casos de testeo se hacen por default con la funcion de activación tangente hiperbólica y un processo incremental a menos que se diga lo contratio.

4.1. Memorización y Generalización: valores de η

En las redes de la *figura 1* y la *figura 4* solo se modificó el η que se utilizó. Observando las figuras notamos que a las 500 epocas la red con un η mayor pudo llegar a un error más chico tanto en generalización como en memorización, mientras que con 1000 epocas la red con el η inferior se mostró superior tanto en ambas métricas, y finalmente al transcurrir 1500 epocas ambas redes tuvieron errores similares. Una observación notable del gráfico de errores resulta ser que los picos de errores en la red con un η mayor son considerablemente más altos.

4.2. Efectos sobre los errores: diferentes arquitecturas

Para medir los efectos de las arquitecturas se comparan varios resultados en los cuales la única variable es la arquitectura.

4.2.1. Efecto del incremento de capas

Sabiendo que las redes de la *figura 1* y la *figura 2* no difieren en ningún parámetro salvo su arquitectura, las utilizamos como punto de comparación. La *figura 1* tiene una sola capa oculta con cincuenta neuronas mientras que la *figura 2* posee dos capas ocultas, la primera capa con treinta neuronas y la segunda con veinte.

El error de generalización y aprendizaje luego de 500, 1000 y 1500 épocas es más bajo en la *figura 2* que la *figura 1*.

4.2.2. Decremento en la cantidad de neuronas de la capa oculta

La *figura 1* presenta una red con cincuenta neuronas en su única capa oculta, mientras que la *figura 3* posee solo diez en su única capa oculta. Utilizando tanto los gráficos como los valores tabulados en la figura podemos ver que la red con una menor cantidad de neuronas en la capa oculta generaliza y aprende peor que una con una mayor cantidad.

4.3. Comparación de funciones de activación

Se utilizó sólo la función de Tangente Exponencial por el dominio del problema a resolver. La función de Sigmóidea Exponencial no corresponde en imagen como salida de la red con los valores de la misma (excede la imagen de $SigExp$).

4.4. Utilización de η *adaptativo*

Comparando tanto la *figura 1* con la *figura 7* como la *figura 2* con la *figura 8* podemos ver que el η adaptativo no muestra una mejora en cuanto a los errores de aproximación ni los errores de generalización.

Se asume que esto se debe a una buena aproximación inicial de $\eta = 0{,}02$, sumado a valores a y b malos o inestables para la función de *Adaptive* η. Valores más chicos de a pueden resultar en cambios más suaves sobre la curva de aprendizaje y hacer que la red sea más conservadora al momento de incorporar conocimiento.

4.5. Aplicación de *momentum*

Para analizar momentum vemos primero las figuras dos y quince en donde se presentan redes de dos capas ocultas con 20 y 30 neuronas respectivamente, en ninguno de los casos se utiliza $\eta = 0.02$. Viendo los errores a 500, 1000 y 1500 épocas se ve que no difieren demasiado, lo que sucede en cambio es que momentum parece producir picos en el error más pronunciados, lo cual se observa claramente en el gráfico.

4.5.1. Efectos del cambio del parametro α

Para ver como el parámetro α afecta a el momentum podemos observar las figuras 15 y 19 en donde el único parametro que se modificó fue el α de 0.9 a 0.5 respectivamente. Se observa como con un α menor el momentum parece funcionar mejor para nuestro terreno ya que produce picos considerablemente menos pronunciados y a su vez un error reducido en las 500, 1000 y 1500 épocas.

4.6. Combinación de *momentum* y η *adaptativo*

Tanto la figura 16 como la 2 poseen la misma distribución de neuronas, dos capas ocultas con 30 y 20 neuronas respectivamente, y utilizan el mismo η inicial, y la única diferencia es que la figura 16 usa una red con momentum y η adaptativo meintras que la 2 no. Comparando las figuras vemos que a 500 épocas el error de generalización y aprendizaje es menor para la red con mejoras que la que no tiene, pero tanto para las 1000 como las 1500 epocas los errores son muy cercanos aunque la red sin mejoras obtiene errores ligeramente menores.

5. Conclusiones

Se selecciona la red 16 por sus características de bajo error y buen nivel de generalización.

5.1. Valor de η

Un η mayor produce mayor variación en el error ya que se le da más potencia a la corrección por gradiente descendiente y esto permite que se pase del mínimo constantemente con mayor potencia que con un eta menor. Sin embargo esta variación más grande en el error resulta buena para llegar rápido a un error bajo. Se eligió satisfactoriamente el valor del η inicial. Como anteriormente fue mencionado, los valores de a y b para η adaptativo no resultan satisfactorios, ya que propician mayores errores para el set de datos analizado.

5.2. Número de Capas Ocultas y Neuronas

Un mayor numero de capas ocultas implica mayor precisión con un número igual de épocas. Sin embargo, se debe considerar que mientras la red de la figura uno poseía 150 sinapsis, la red de la figura dos contiene 670, un poco más del cuádruple, lo que llevaría a interpretar que un mayor numero de conexiones favorece la precisión del aprendizaje. Lo que podemos concluir con confianza es que para nuestro problema la red de la figura dos es mejor que la de la figura uno bajo las condiciones que se la analizó.

5.3. Poder de Generalización de la Red

Incrementar la cantidad de neuronas en la capa oculta aumenta la capacidad de generalizar y memorizar de la red.

Como el α hace que en una meseta el learning rate efectivo sea

$$\frac{\eta}{1-\alpha}$$

podemos ver cómo si las mecetas se producen cerca del minimo de error, un alpha mas cercano a uno produce variaciones en los pesos mayores.

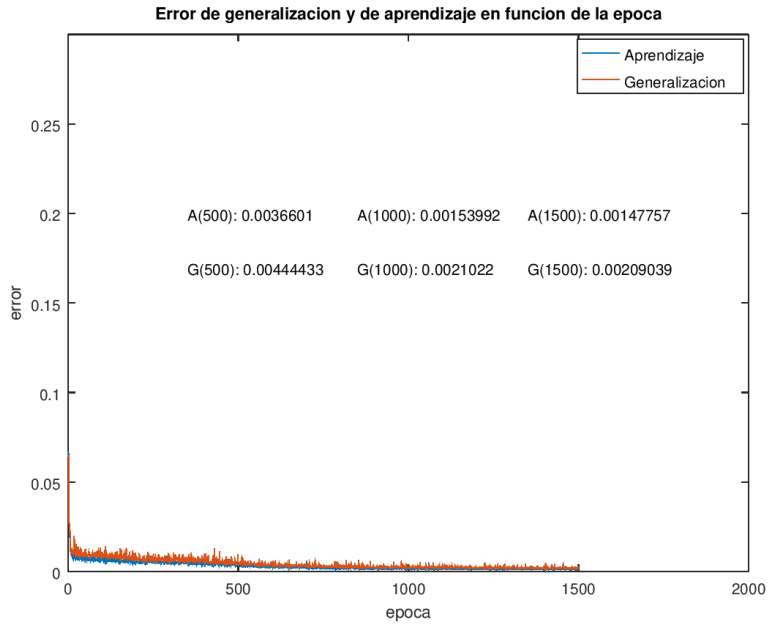
Referencias

Million, E. (2007). *http://buzzard.ups.edu/courses/2007spring/projects/million-paper.pdf.*

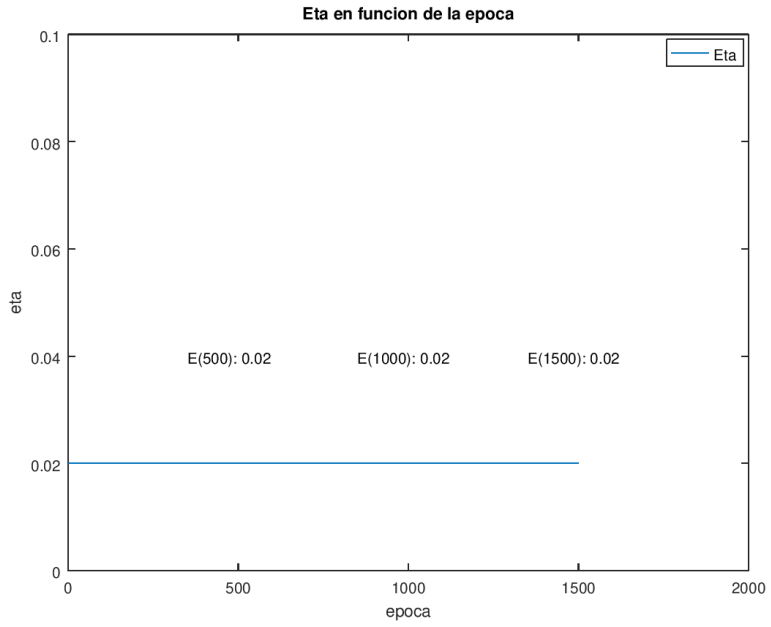
Apéndice

Figura	η	η Adaptativo	Momentum	Arquitectura	Función g
1	0.02	No	No	2, 50, 1	tanh
2	0.02	No	No	2, 30, 20, 1	tanh
3	0.02	No	No	2, 10, 1	tanh
4	0.04	No	No	2, 50, 1	tanh
5	0.04	No	No	2, 30, 20, 1	tanh
6	0.04	No	No	2, 10, 1	tanh
7	0.02	Sí	No	2, 50, 1	tanh
8	0.02	Sí	No	2, 30, 20, 1	tanh
9	0.02	Sí	No	2, 10, 1	tanh
10	0.04	Sí	No	2, 50, 1	tanh
11	0.04	Sí	No	2, 30, 20, 1	tanh
12	0.04	Sí	No	2, 10, 1	tanh
13	0.02	No	Sí	2, 50, 1	tanh
14	0.02	Sí	Sí	2, 50, 1	tanh
15	0.02	No	Sí	2, 30, 20, 1	tanh
16	0.02	Sí	Sí	2, 30, 20, 1	tanh
17	0.02	No	Sí	2, 50, 1	tanh
18	0.02	Sí	Sí	2, 50, 1	tanh
19	0.02	No	Sí	2, 30, 20, 1	tanh
20	0.02	Sí	Sí	2, 30, 20, 1	tanh

Cuadro 1: Tabla de Gráficos Según Parámetros

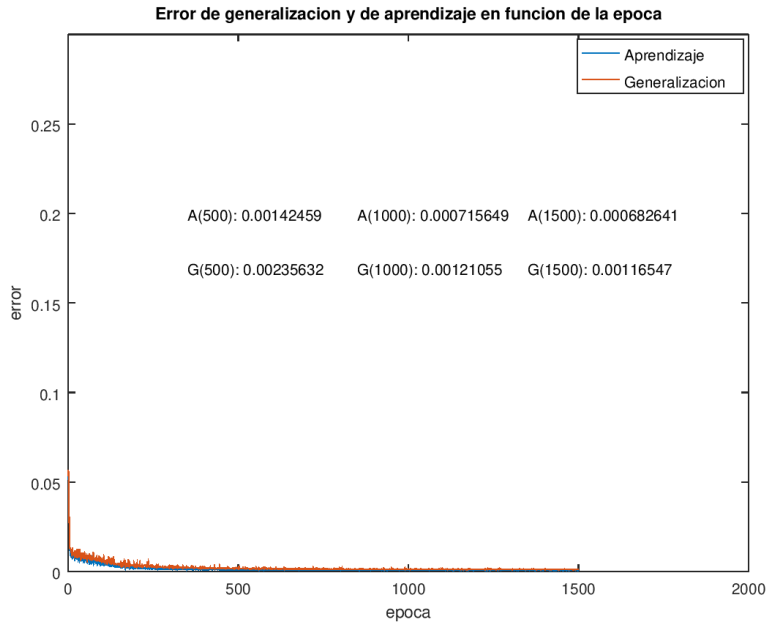


(a) Errores

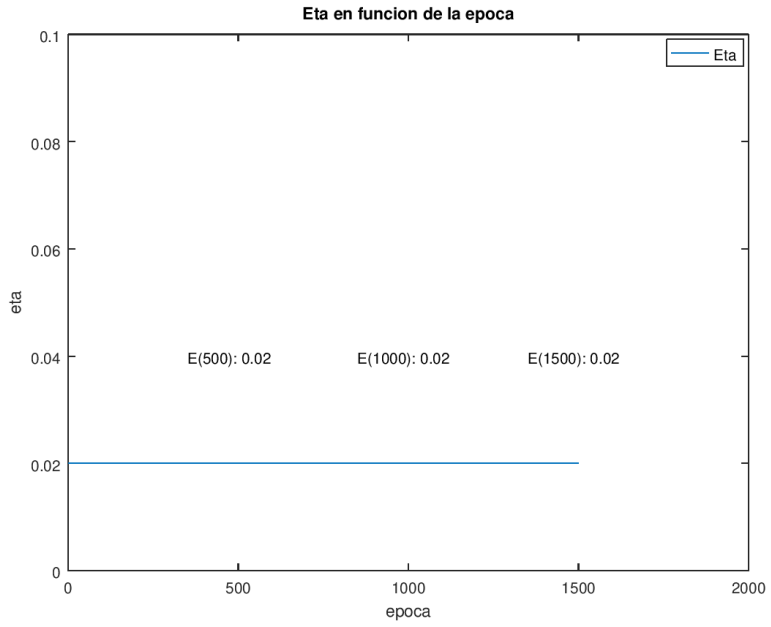


(b) Valor de η

Figura 1: Resultado para el caso 1: capa oculta de 50 neuronas, $\eta = 0{,}02$

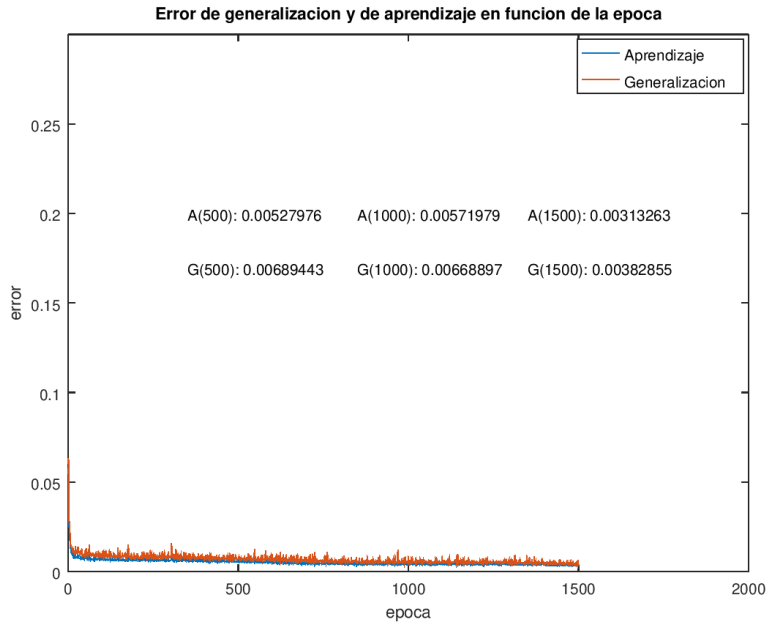


(a) Errores

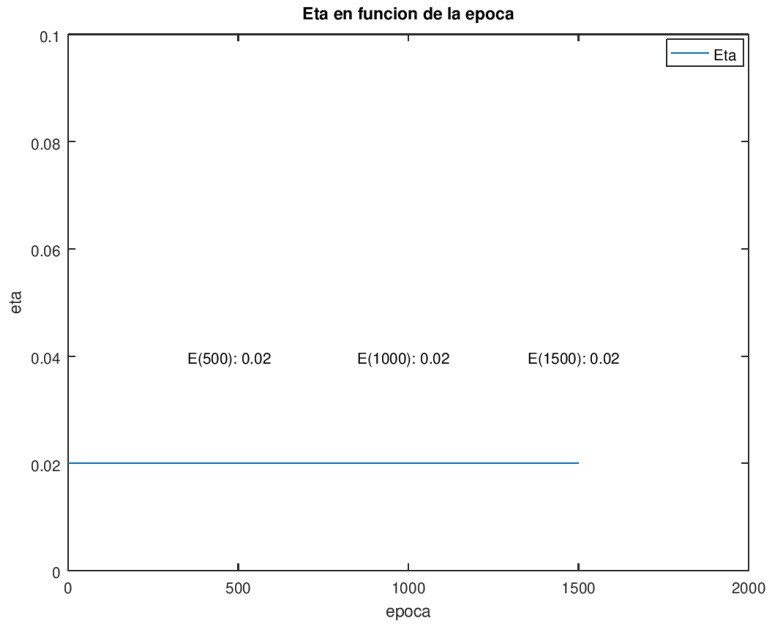


(b) Valor de η

Figura 2: Resultado para el caso 2: capas ocultas de 30, 20 neuronas, $\eta = 0{,}02$

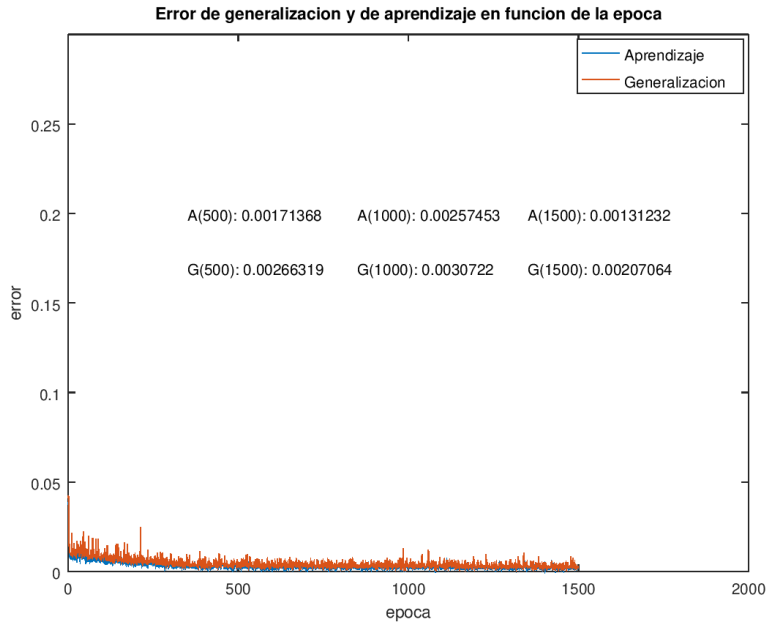


(a) Errores

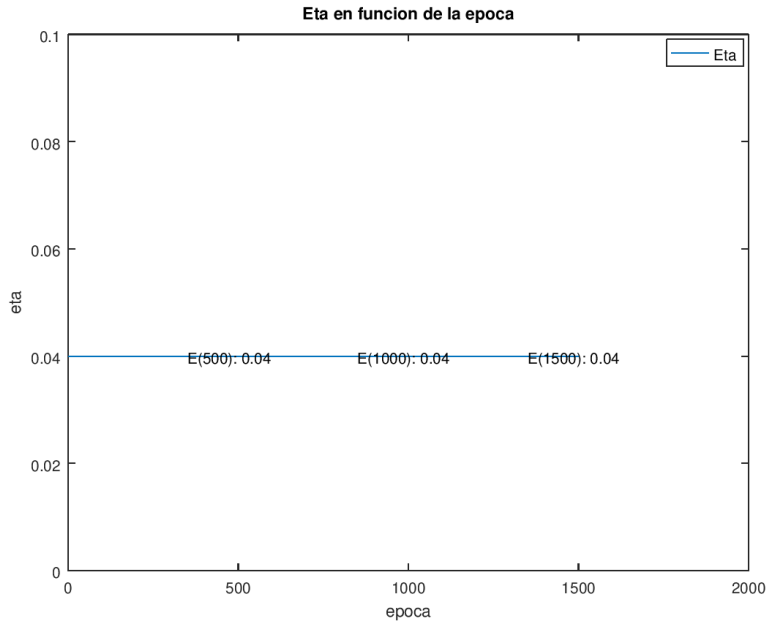


(b) Valor de η

Figura 3: Resultado para el caso 3: capa oculta de 10 neuronas, $\eta = 0{,}02$

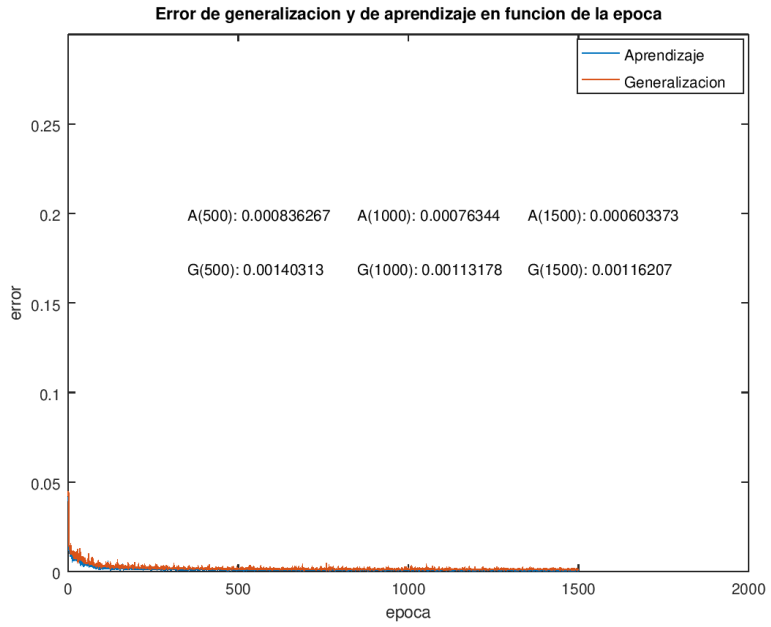


(a) Errores

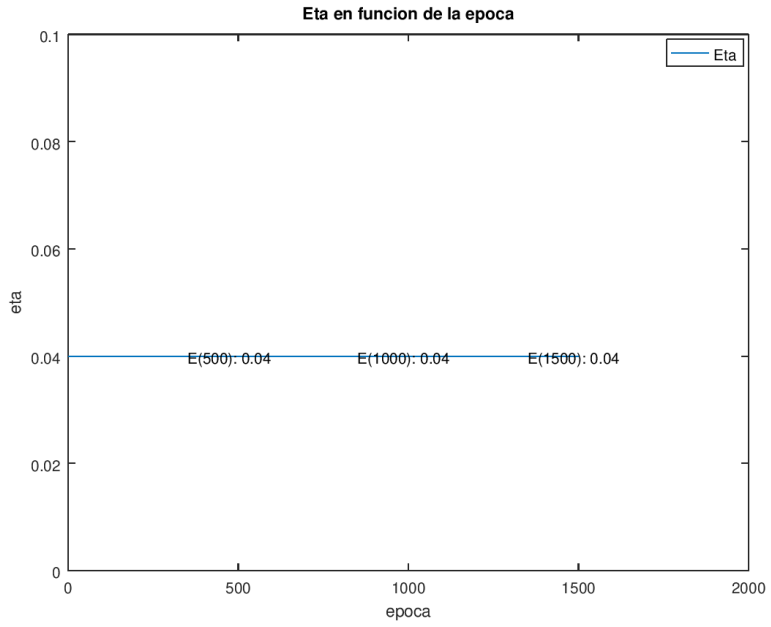


(b) Valor de η

Figura 4: Resultado para el caso 4: capa oculta de 50 neuronas, $\eta = 0{,}04$

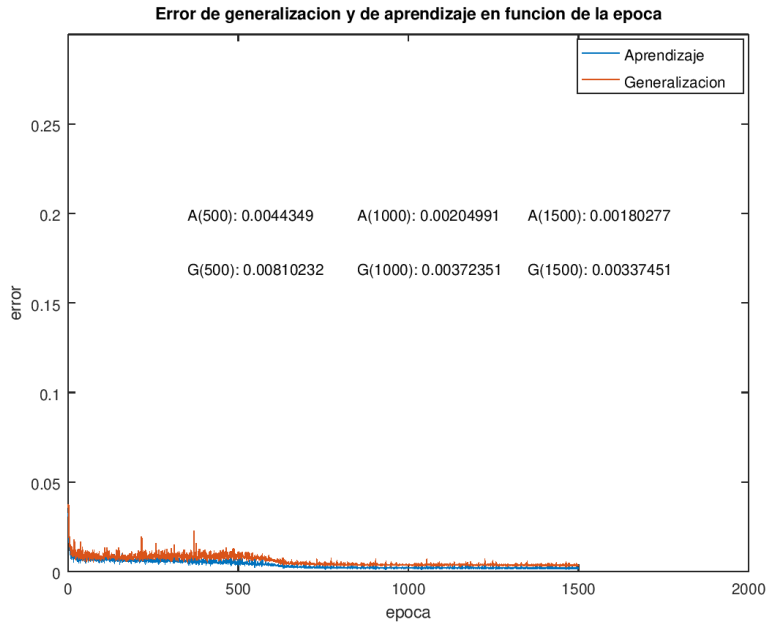


(a) Errores

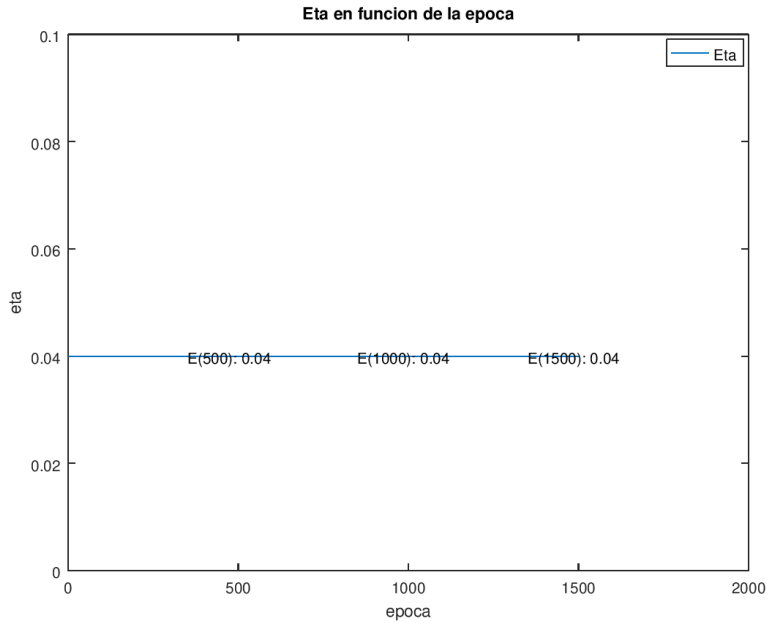


(b) Valor de η

Figura 5: Resultado para el caso 5

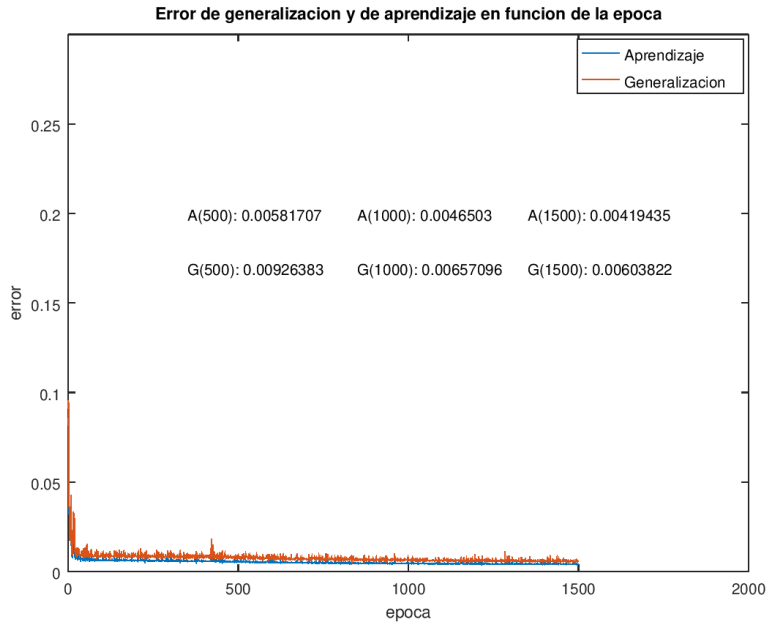


(a) Errores

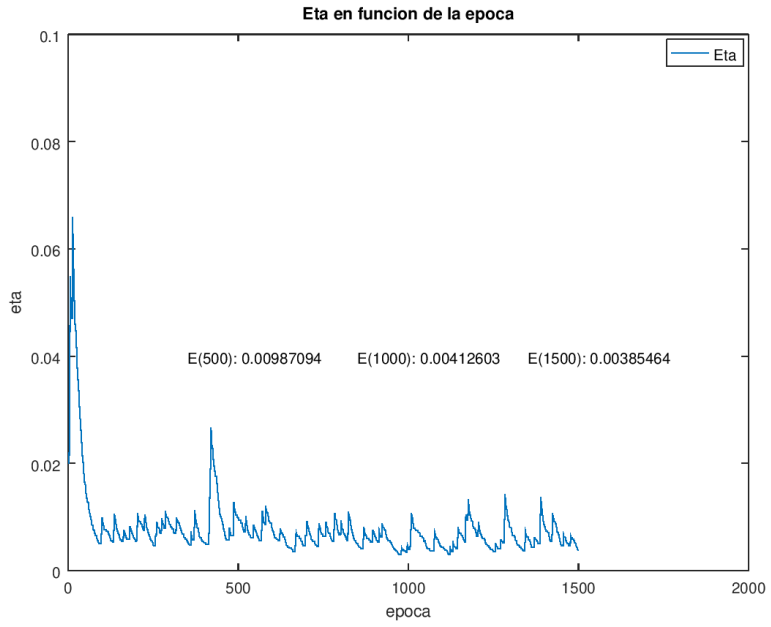


(b) Valor de η

Figura 6: Resultado para el caso 6

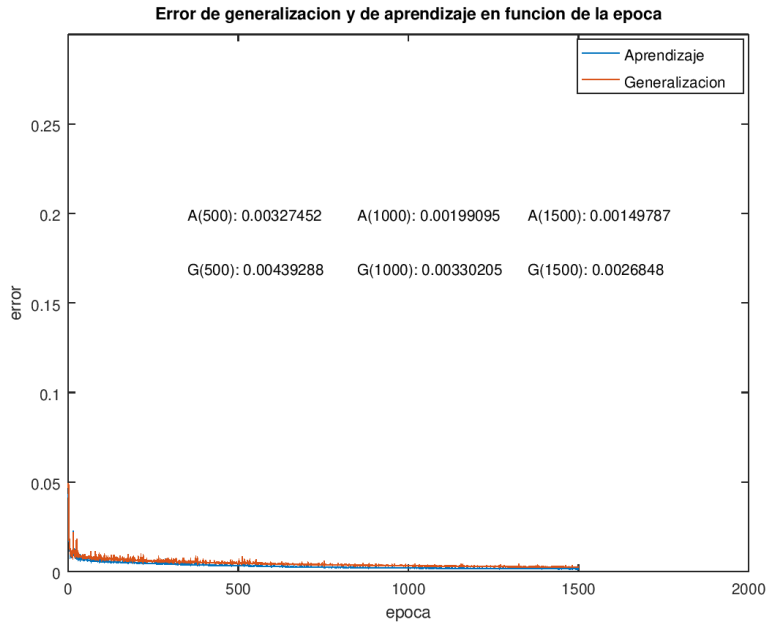


(a) Errores

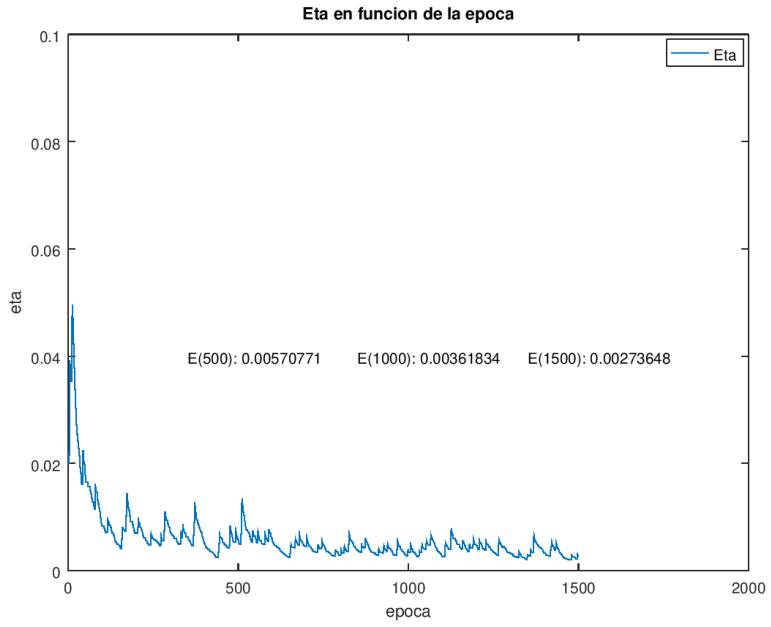


(b) Valor de η

Figura 7: Resultado para el caso 7

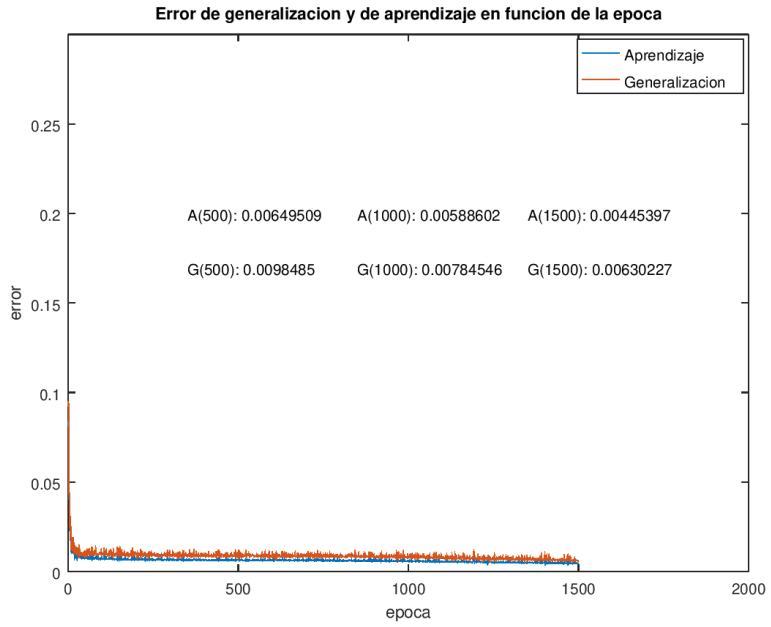


(a) Errores

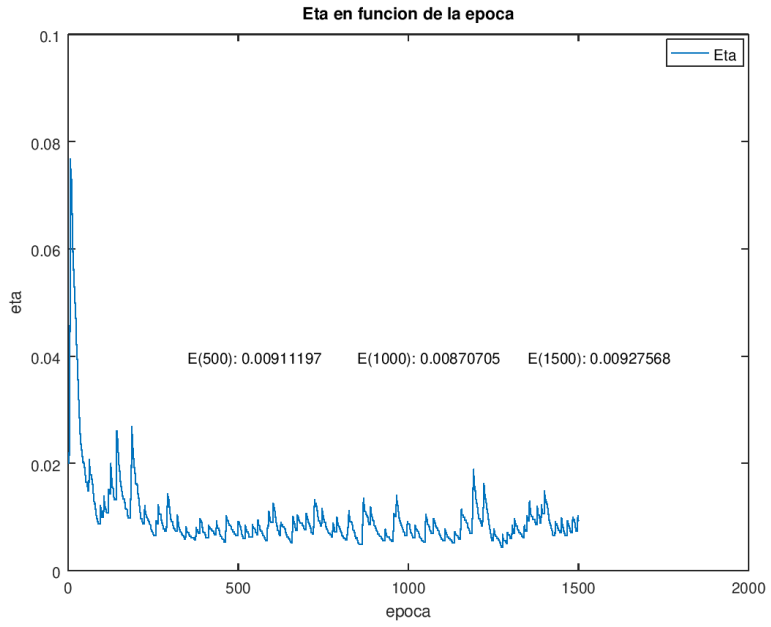


(b) Valor de η

Figura 8: Resultado para el caso 8

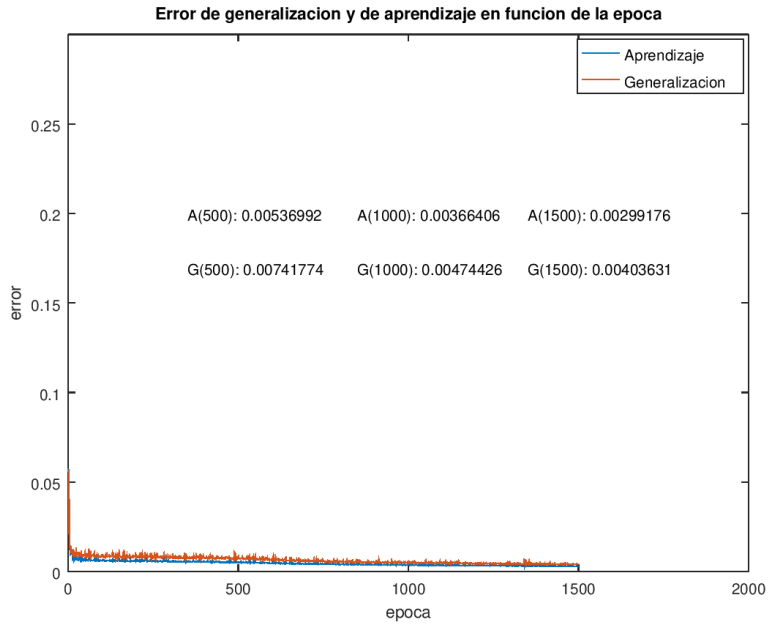


(a) Errores

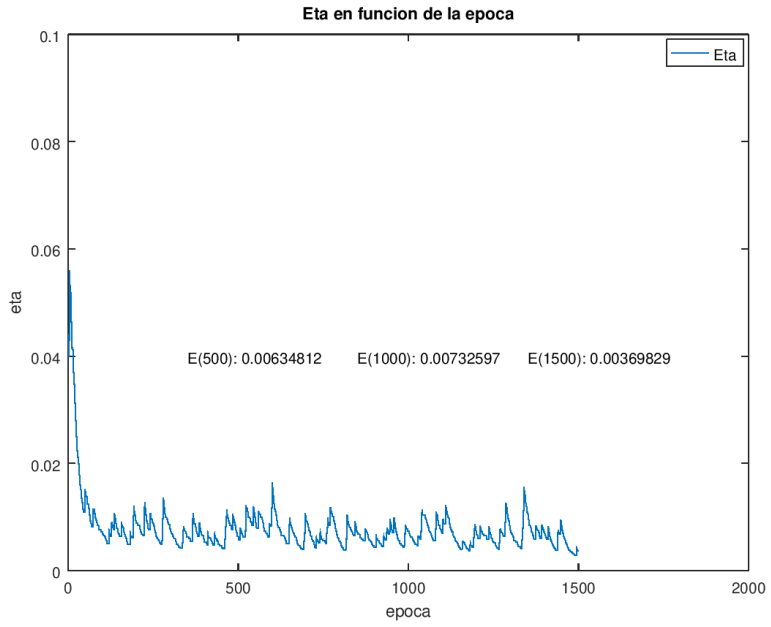


(b) Valor de η

Figura 9: Resultado para el caso 9

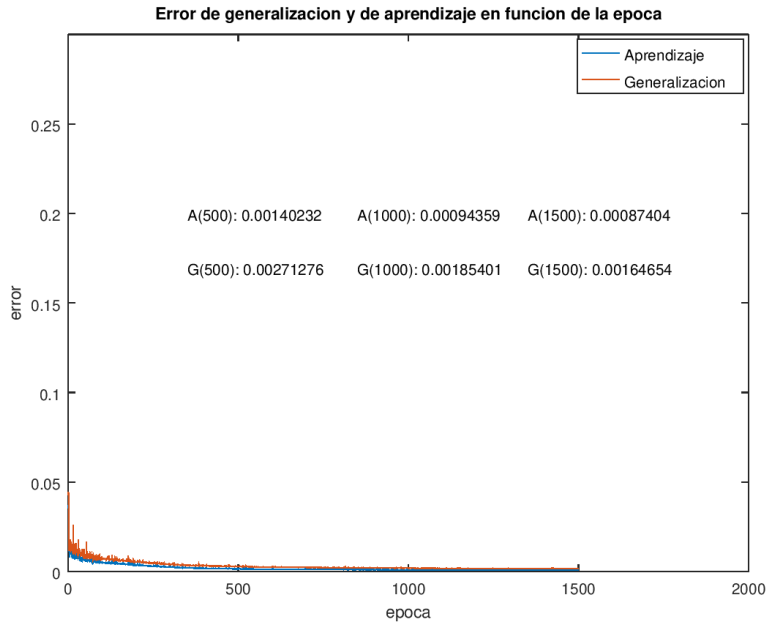


(a) Errores

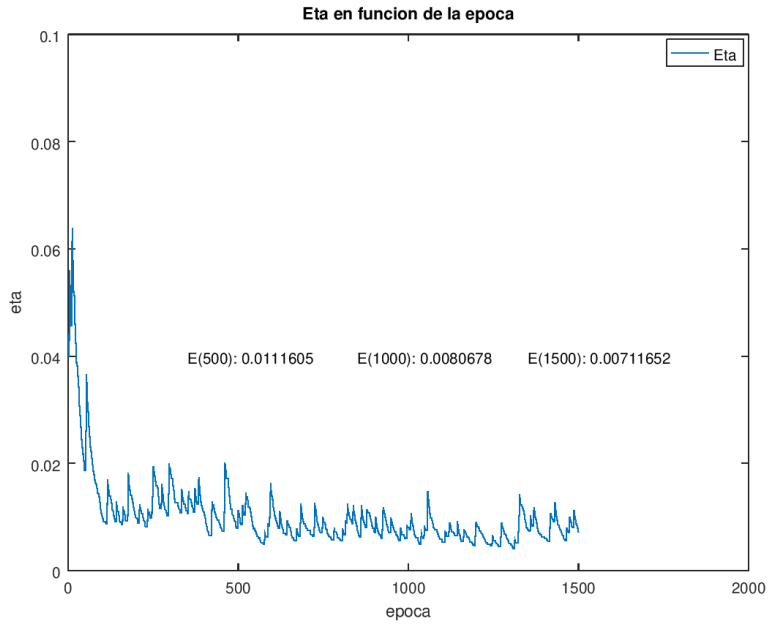


(b) Valor de η

Figura 10: Resultado para el caso 10

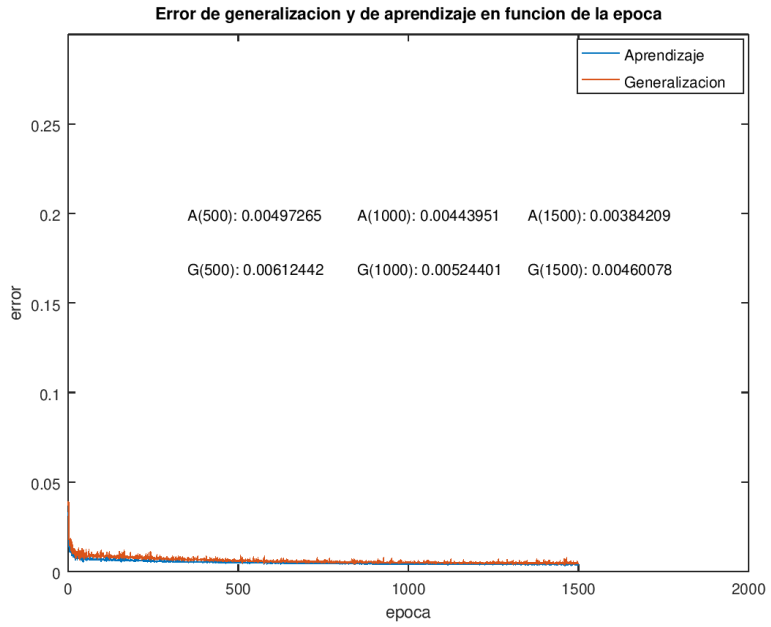


(a) Errores

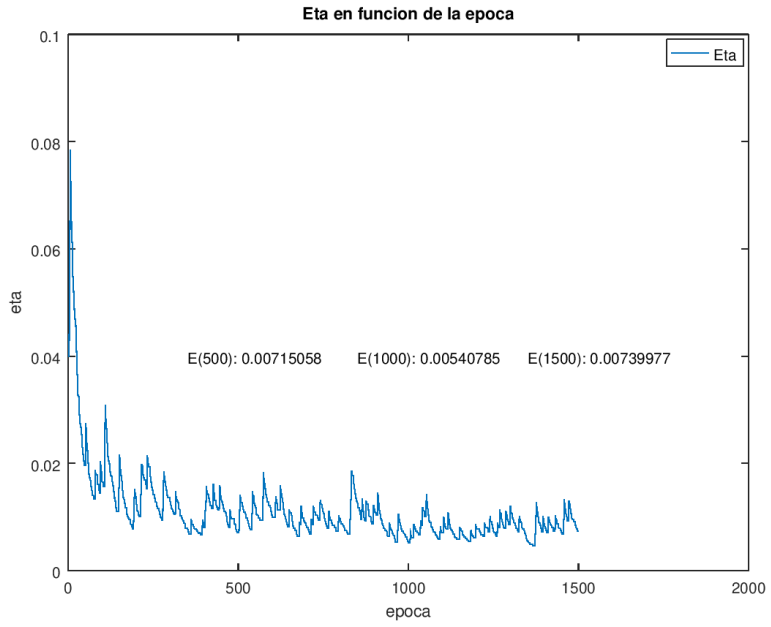


(b) Valor de η

Figura 11: Resultado para el caso 11

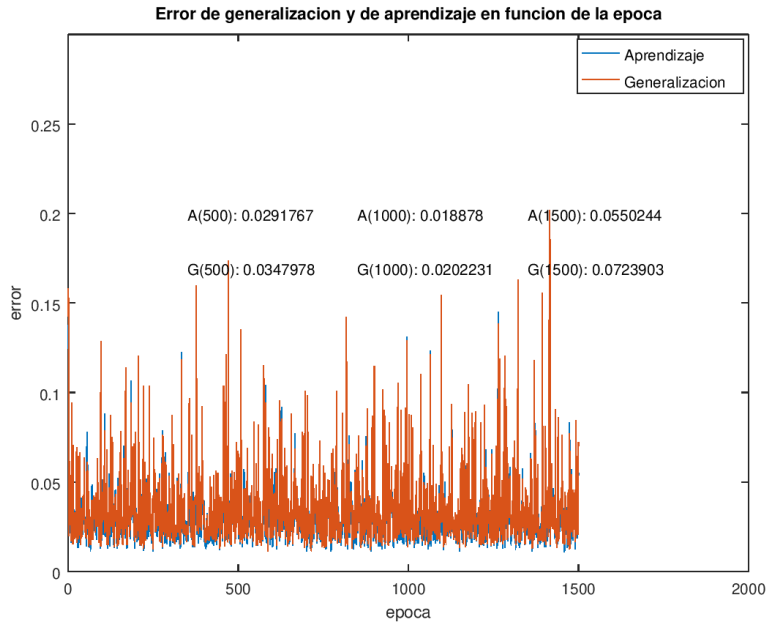


(a) Errores

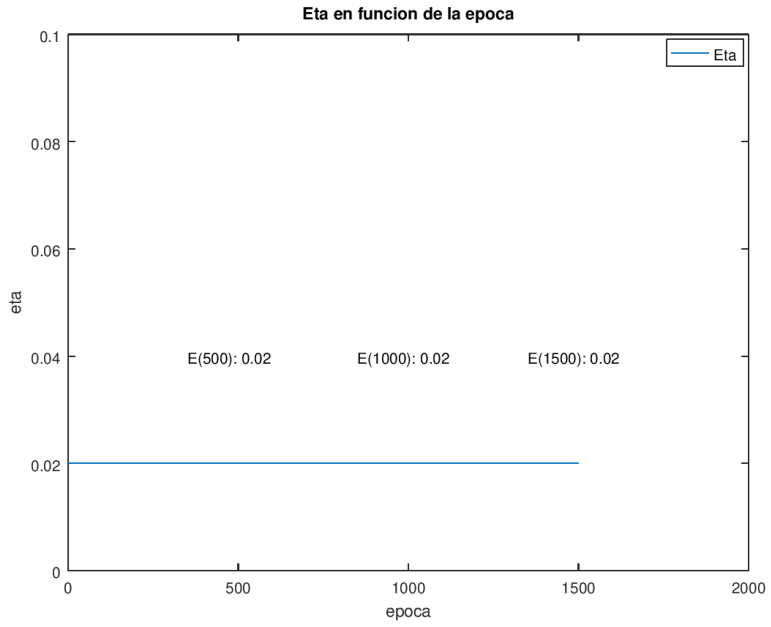


(b) Valor de η

Figura 12: Resultado para el caso 12

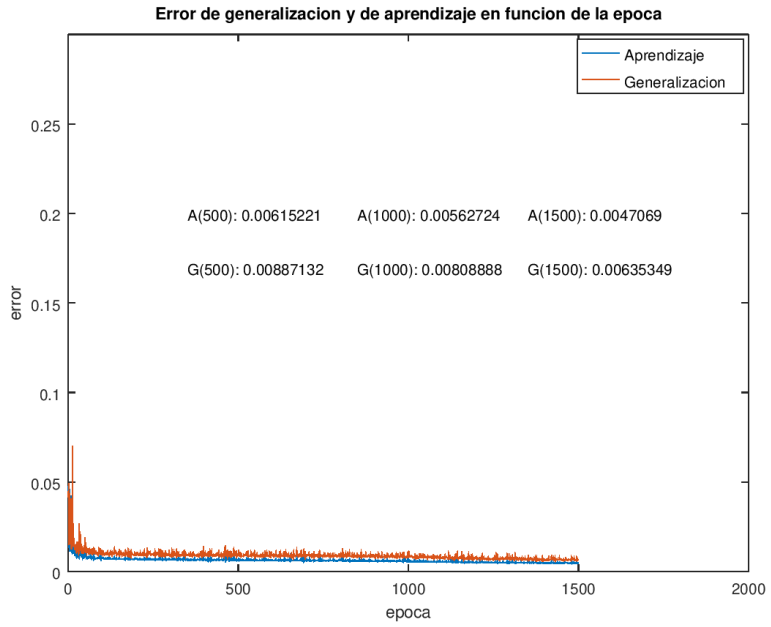


(a) Errores

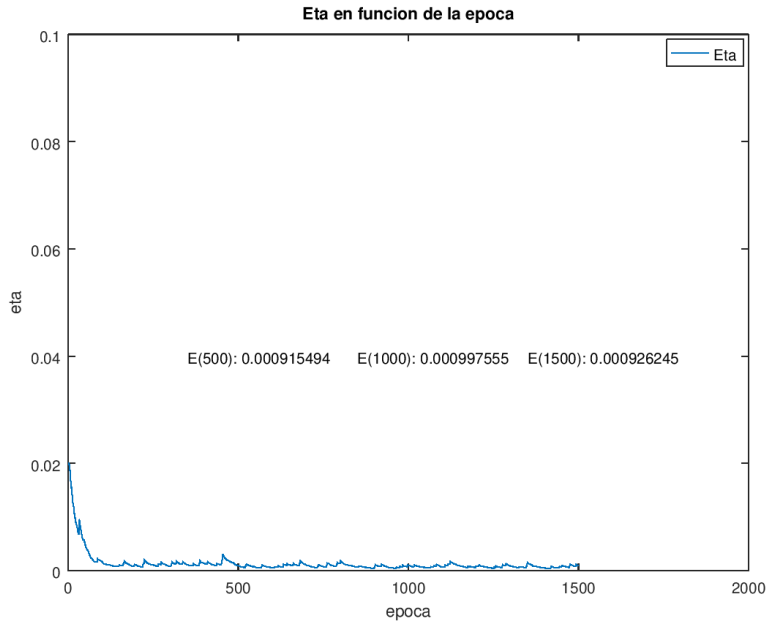


(b) Valor de η

Figura 13: Resultado para el caso 13

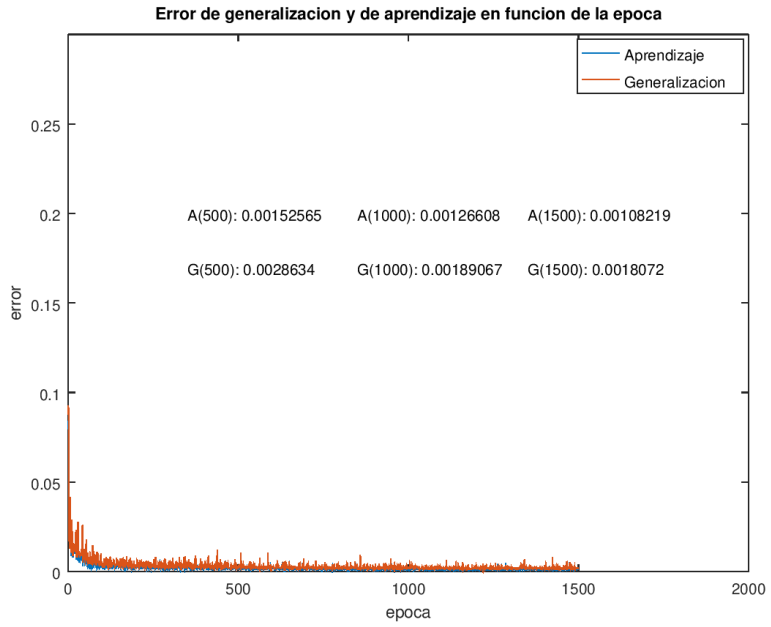


(a) Errores

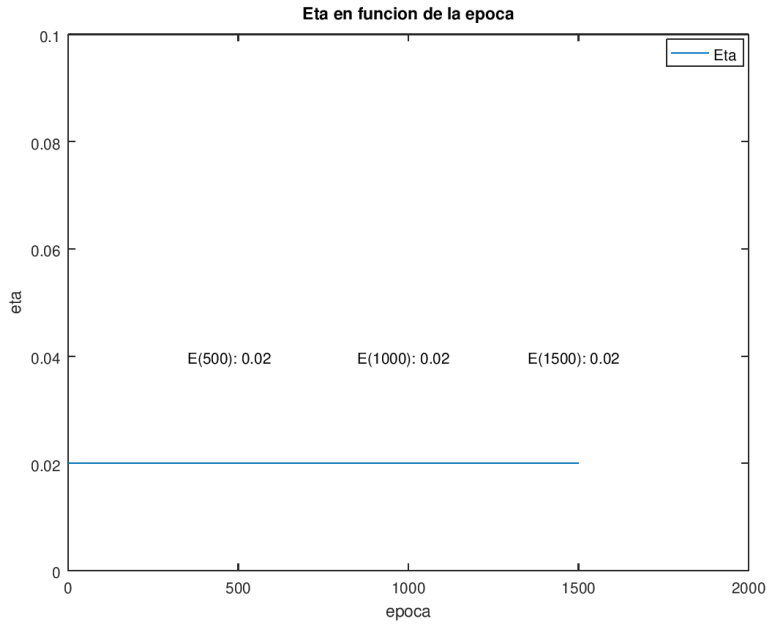


(b) Valor de η

Figura 14: Resultado para el caso 14

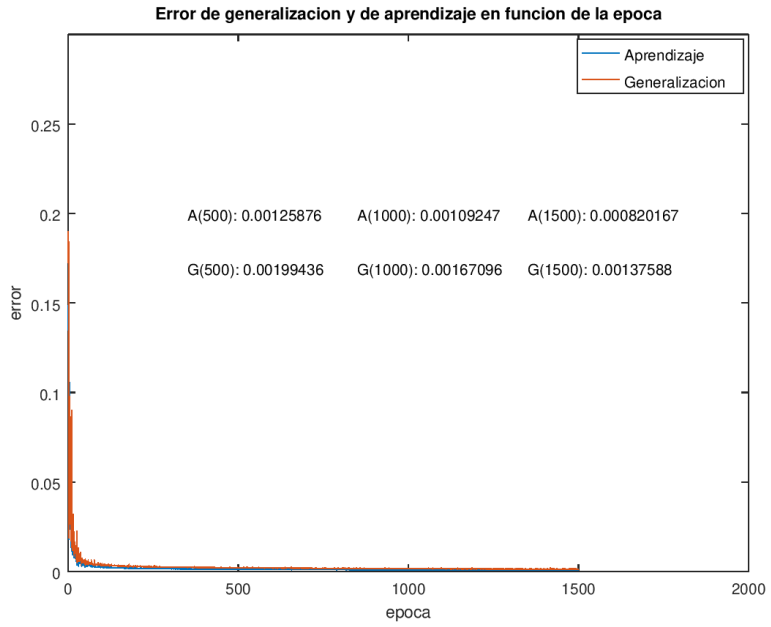


(a) Errores

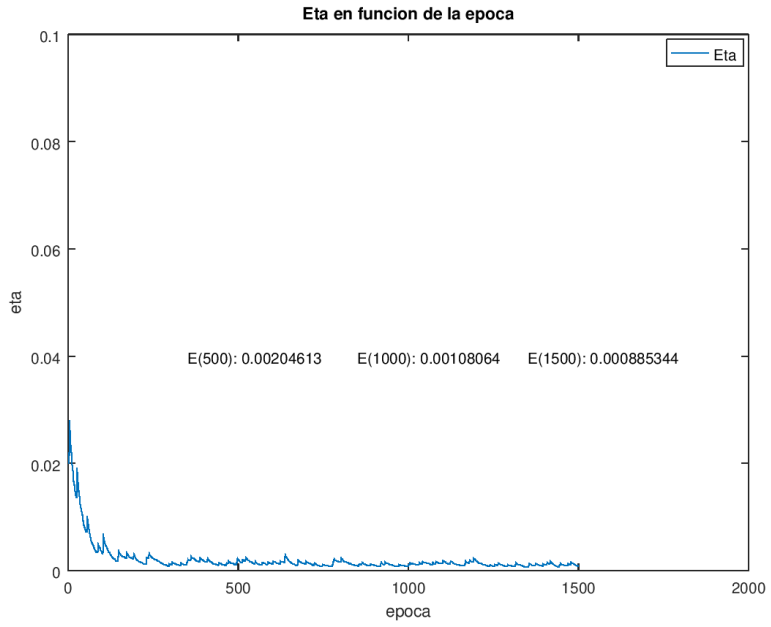


(b) Valor de η

Figura 15: Resultado para el caso 15

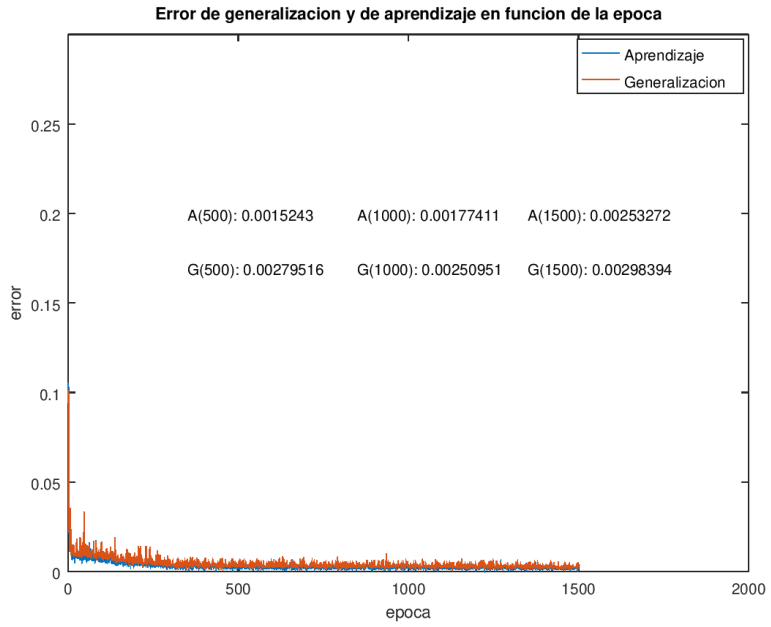


(a) Errores

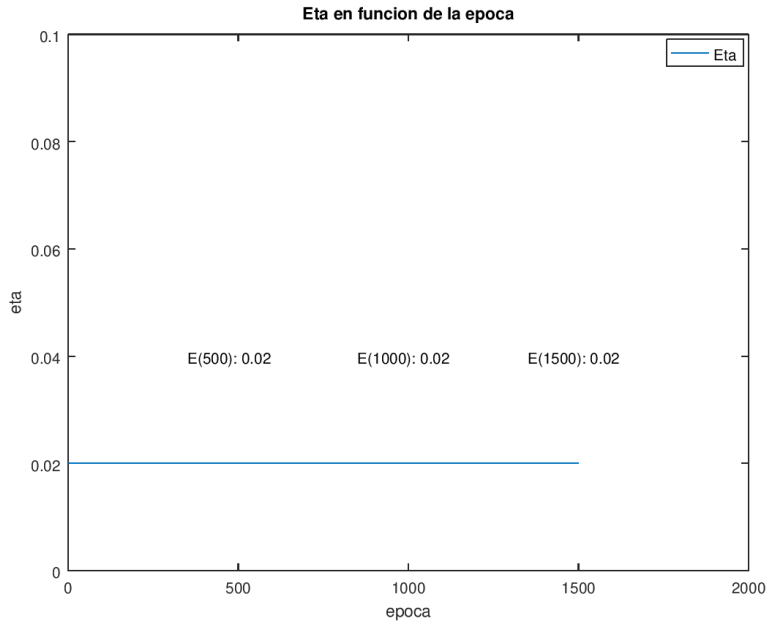


(b) Valor de η

Figura 16: Resultado para el caso 16

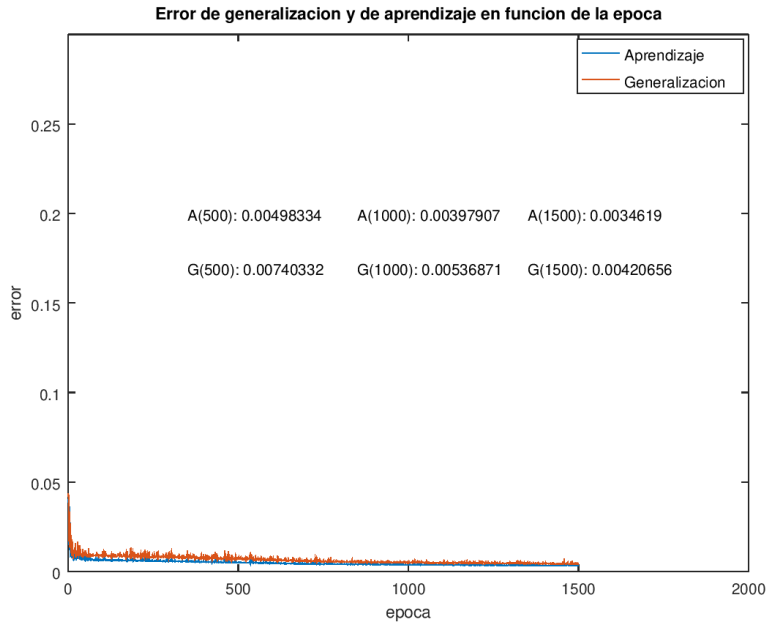


(a) Errores

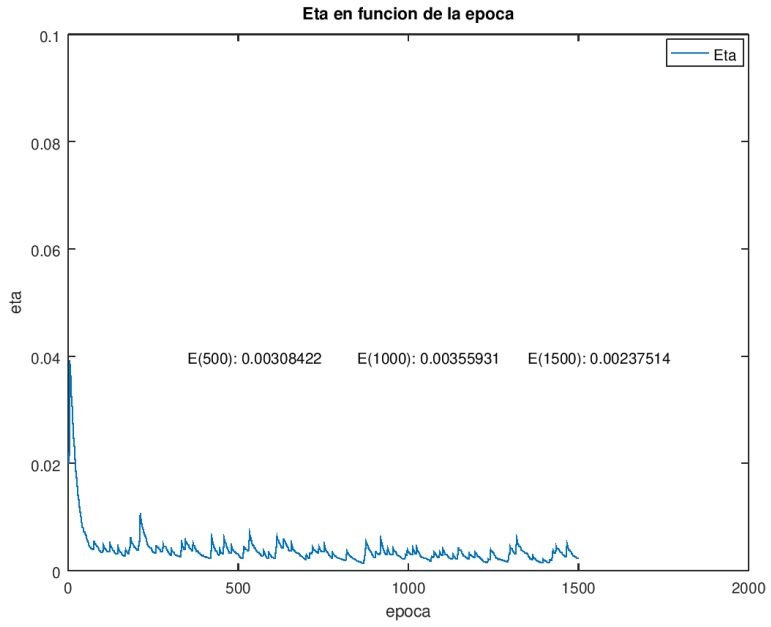


(b) Valor de η

Figura 17: Resultado para el caso 17

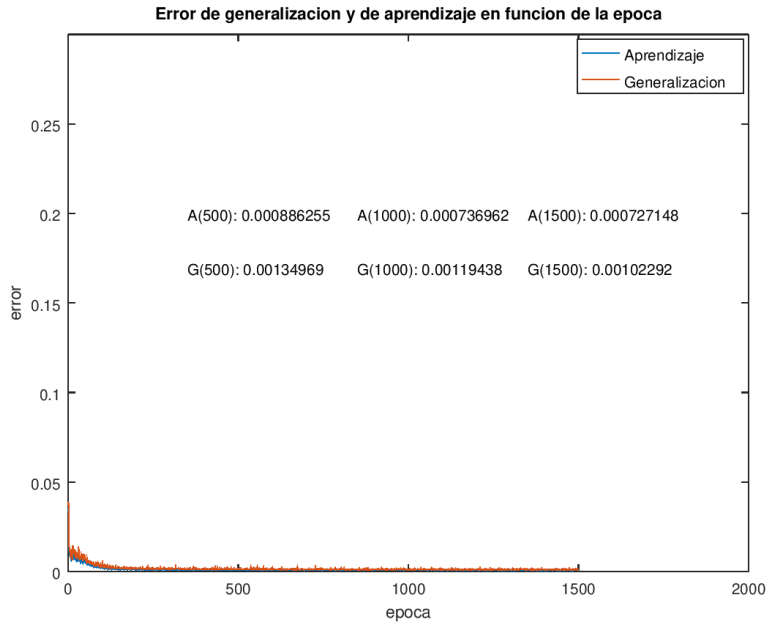


(a) Errores

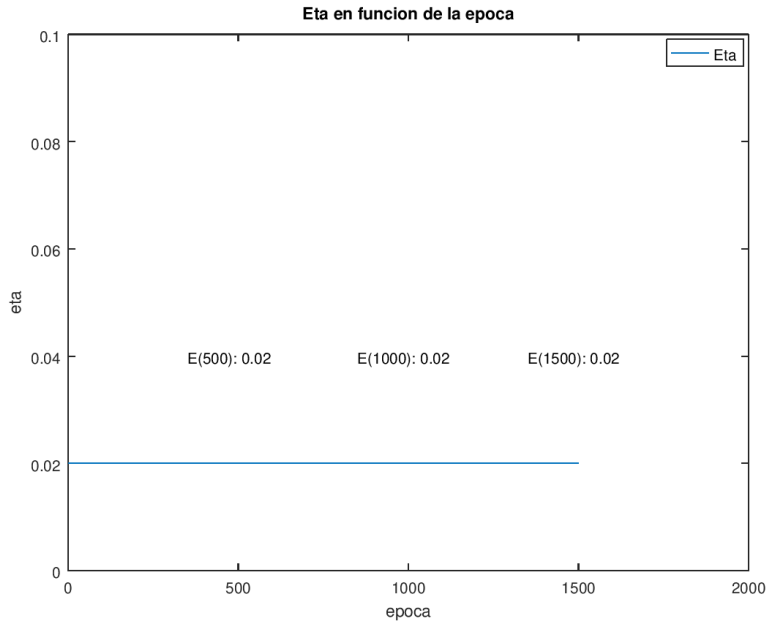


(b) Valor de η

Figura 18: Resultado para el caso 18

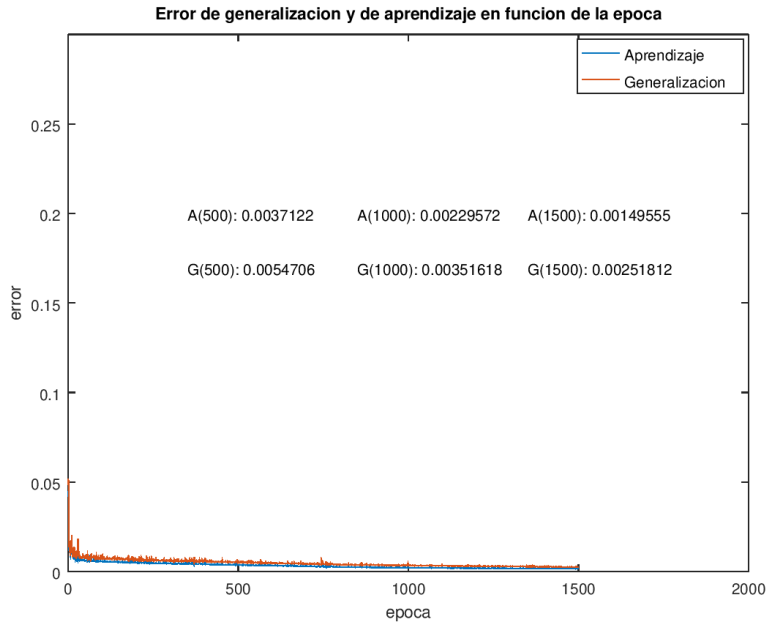


(a) Errores

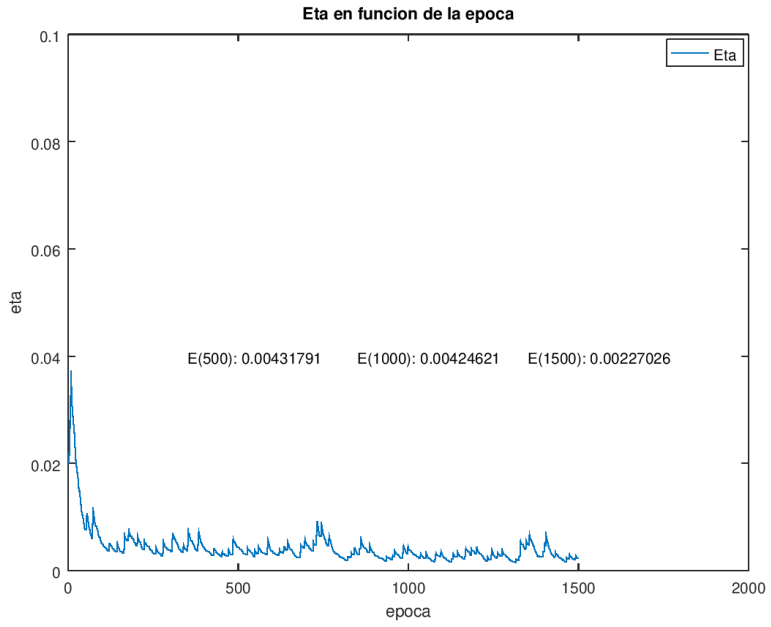


(b) Valor de η

Figura 19: Resultado para el caso 19



(a) Errores



(b) Valor de η

Figura 20: Resultado para el caso 20