DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 1

5/5/2015 Redes Neuronales Artificiales

Integrante	LU	Correo electrónico
Landini, Federico Nicolás	034/11	federico91_fnl@yahoo.com.ar
Rama Vilariño, Roberto Alejandro	490/11	bertoski@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

1. Introducción

En el presente trabajo se busca atacar dos problemas a partir del uso de redes neuronales artificiales [1]. Se nos presentaron dos problemas con características diferentes, los cuales se busca evaluar el rendimiento de esta técnica y su versatilidad.

El primero consiste en diagnosticar tumores como benignos o malignos a partir de ciertas características recogidas de muestras de células, por lo que puede ser catalogado como un problema de clasificación. El segundo consiste en determinar los requerimientos de carga energética para calefaccionar y refrigerar edificios en función de ciertas características de los mismos, lo que puede ser catalogado como un problema de regresión.

Para resolver los problemas consideramos distintas configuraciones de redes para cada problema. Luego, evaluamos su rendimiento con el fin de comparar los resultados y elegir la configuración que mejor se adaptaba a los conjuntos de datos, pero a la vez manteniendo generalidad a fin de evitar el problema de *overfitting*.

Durante el testeo decidimos utilizar la técnica de cross-validation [2], con el fin de realizar un analisis mas robusto respecto al rendimiento. Esta tecnica nos permitio tener una idea más real del nivel de generalidad de los resultados obtenidos con las redes.

2. Desarrollo

2.1. Implementacion de la red neuronal

La implementación de la red neuronal se realizó en MatLab. Decidimos que era el entorno más apropiado para el trabajo, ya que por tratarse de un trabajo de experimentación no estábamos buscando performance y nos brinda diversas comodidades en operaciones con matrices.

Se utilizó la implementación de la red neuronal como se presentó en clase, y se agrupó cada parte de la misma en una clase definida en MatLab, bajo el nombre de **MyMultiPerceptron**. Puntualmente aplicamos aprendizaje incremental para el entrenamiento e implementamos la modificación del **momentum** para que la red pueda converger más rápidamente. Tanto el momentum (α) como el learning rate (γ) no se modifican durante el entrenamiento.

A modo de dejar bien asentada la implementación, la presentamos a continuación en formato de pseudocódigo. Sin embargo, no daremos mayores explicaciones ya que no presenta ninguna innovación. La función de activación fue reemplazada correspondientemente según el modo en el que la red neuronal estuviese corriendo, daremos más detalles en la sección que especifica sus opciones.

function FEEDFORWARD(w, x)
  y = propagateFeed(w, x)
  return $y_{|y|}$
end function

function PROPAGATEFEED(w, x)
  $y_1 = x$
  for $i = [1..|\text{w}|]$ **do**
    y_{i+1} =activation($[y_i\ 1] * \text{w}_i$)
  end for
  return y
end function

function TRAIN(w, data, err_{min}, ep_{max}, γ, α)
  $\Delta lw = \emptyset$
  while $e\ > \text{err}_{min} \wedge t\ < \text{ep}_{max}$ **do**
    $e = 0$
    $t = t + 1$
    for $x, z = [1..|\text{data}|]$ **do**
      y = propagateFeed(x)
      $[fe\ \Delta w]$ = correction(w, y, z, γ)
      w = adaptation(w, Δw, Δlw, α)
      $\Delta lw = \Delta w$
      $e = e + fe$
    end for
  end while
end function

function CORRECTION(w, y, z, γ)
  $\Delta w = \emptyset$
  $re = z - y_{|y|}$
  $fe = ||re||_1/|re|$
  for $i = [|w|.,1]$ **do**
    $re = re$.* activation'(y_{i+1})
    $\Delta w_i = \gamma\ *\ [y_i\ 1]' * re$
    $re = re * \text{w}_{[1..|w|-1]}$
  end for
  return $[fe\ \Delta w]$
end function

function ADAPTATION(w, Δw, Δlw, α)
  for $i = [1..|w|]$ **do**
    $\text{w}_i = \text{w}_i + \Delta w + \alpha * \Delta lw$
  end for
  return w
end function

2.2. Preprocesamiento de los datos

A fin de lograr que todas las características observadas tengan la misma influencia en la red (y evitar que la influencia de alguna característica con amplitud y valor absoluto en sus valores sea mayor que la de una con análogos menores), analizamos los resultados al hacer un preprocesamiento de cada caraterística en las bases de datos.

2.3. Generación de instancias de prueba

Con el objetivo de evaluar diferentes arquitecturas de redes decidimos particionar el conjunto de datos disponibles (realizando la misma operación para cada dataset) utilizando k-fold cross-validation para disminuir los efectos de tomar conjuntos poco representativos y analizar el funcionamiento de una determinada selección de parámetros sobre distintas particiones de los datos. Sin embargo, considerando que debemos realizar una serie de pruebas en un tiempo acotado, decidimos acotarlas de la siguiente manera.

Particionamos la totalidad de los datos en cuatro partes, cada una considerando el 25 % de los mismos. Dada esa partición generamos 4 folds diferentes. Así, entrenamos con el 75 % y testeamos en el 25 % restante con cada una de las variantes.

2.4. Variantes consideradas

El desarrollo del trabajo consistió fundamentalmente en analizar distintas variantes para la red a fin de obtener parámetros que permitiesen obtener buenos resultados en cada problema. Para ello, cada uno de los mismos fue tratado de manera independiente y se realizaron distintas pruebas que se detallan a continuación.

2.4.1. Arquitectura adecuada al problema

Uno de los aspectos fundamentales sobre el funcionamiento de una red neuronal es su configuración respecto de la cantidad de capas y de neuronas en cada una de ellas. Durante el desarrollo consideramos distintas configuraciones, pero dado que los problemas son simples una única capa oculta debería ser suficiente dado que esto es cierto para la mayoria de los problemas[3]. Finalmente nos queda elegir la cantidad de neuronas en dicha capa.

Para tener un punto de referencia sobre el cual comenzar a evaluar los rendimientos de las arquitecturas consideramos una heurística que nos sugiere elegir una cantidad de neuronas en la capa oculta igual a la cantidad de entradas más la cantidad de salidas sobre dos[3].

En el caso del conjunto de datos del problema 1, dado que hay 30 entradas y 1 salida, evaluamos arquitecturas donde variamos las neuronas de la capa oculta en el conjunto $\{5, 10, 15, 20, 25\}$.

Para el problema 2, dado que hay 8 entradas y 2 salidas, evaluamos una capa oculta con cantidades en el conjunto $\{2, 4, 6, 8, 10, 12, 14\}$.

2.4.2. Épocas necesarias para obtener un buen resultados

Otro punto clave en el entrenamiento de la red es la cantidad de épocas a utilizar. Para evaluar este aspecto consideramos distintas cantidades para cada una de las arquitecturas mencionadas previamente. Los maximos de las mismas fueron elegidas a partir de que observamos que la red convergia suficientemente bien en dichos valores. El momentum durante estas pruebas se encontraba desactivado ($\alpha = 0$).

Para el problema 1, entrenamos usando $10, 20, 30, 40, 50, 60, 70, 80, 90, 100$ épocas y para el problema 2 fueron $500, 1000, 1500, 2000$.

2.4.3. Utilización de distintos valores de learning rate

Una vez escojidas arquitecturas y cantidades de iteraciones que permitieron obtener buenos resultados realizamos variaciones en el learning rate. Realizamos evaluaciones utilizando los valores $\gamma = 0{,}1, 0{,}2, 0{,}3, 0{,}4, 0{,}5$.

2.4.4. Normalización de los datos

En el primer dataset solamente era posible normalizar los datos de entrada ya que la salida corresponde a un problema de clasificación. Sin embargo, en el segundo caso, donde se trata de un problema regresión, la salida puede normalizarse para el entrenamiento, lo que puede producir

cambios significativos en la red. Evaluamos este punto para corroborar si se encontraban diferencias significativas.

2.4.5. Momentum

Una vez elegida la arquitectura a utilizar, evaluamos la misma fijando una cantidad de épocas suficientes para asegurar su convergencia y variando el momentum para examinar sus efectos. Los valores elegidos para evaluar este punto fueron los contenidos en el conjunto

$$\alpha = \{i * 0{,}05 \mid i \in [1, 19] \subset N\}$$

3. Resultados

Uno de los primeros experimentos realizados fue variar la cantidad de neuronas en una sola capa oculta para cada problema considerando las cantidades de épocas mencionadas anteriormente. A continuación presentamos resultados sobre dichos experimentos.

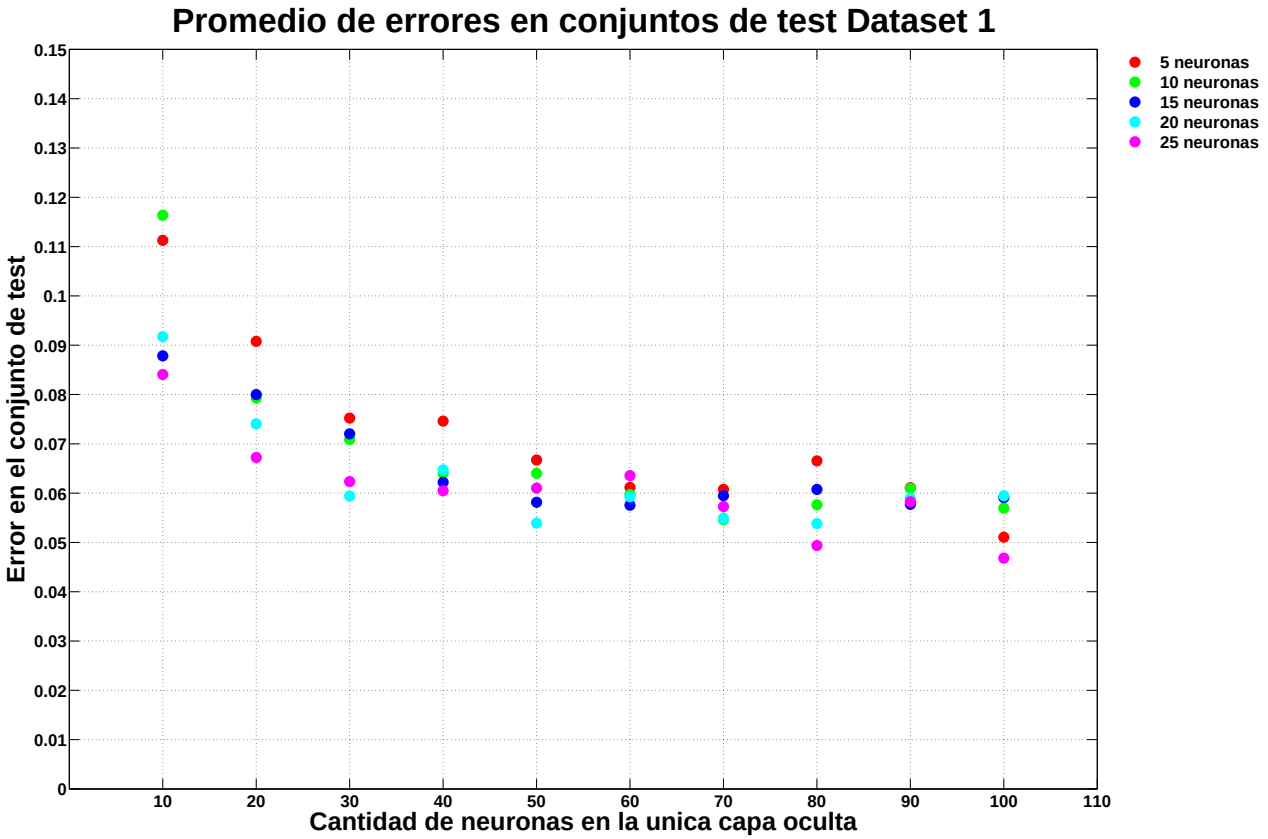


Figura 1: Promedio de error entre los 4 folds para el dataset 1 calculado sobre el conjunto de test usando $\gamma = 0{,}1$

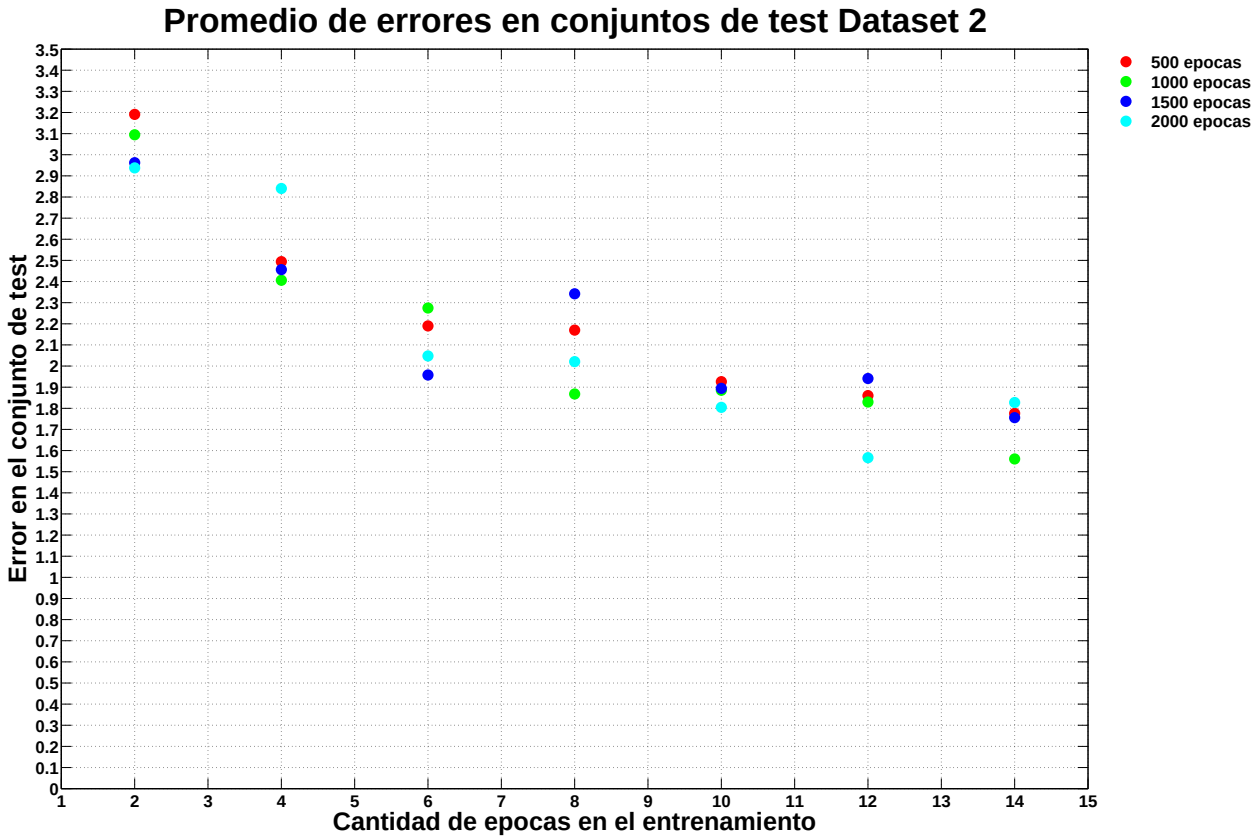
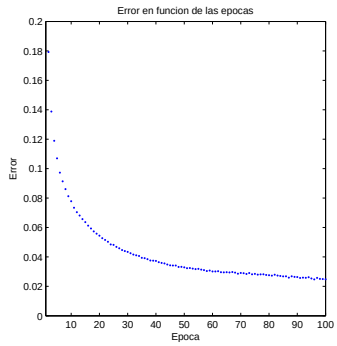


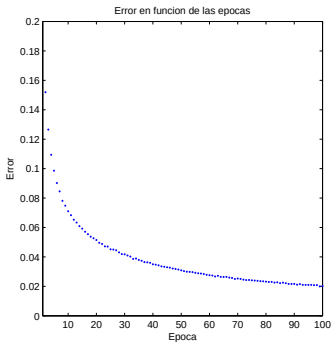
Figura 2: Promedio de error entre los 4 folds para el dataset 2 calculado sobre el conjunto de test usando $\gamma = 0{,}1$

3.1. Problema 1

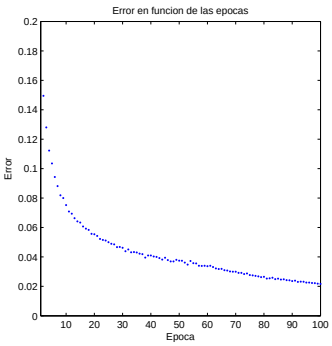
Dado que según la Figura 1 las configuraciones con 5 y 10 neuronas no presentan tan buenos resultados, mostramos a continuación (Figura 3) gráficos sobre la convergencia del error en función de las épocas usando las otras arquitecturas evaluadas. Para ello mostramos el fold 1 pero los demás presentan un comportamiento similar.



(a) Usando 15 neuronas.



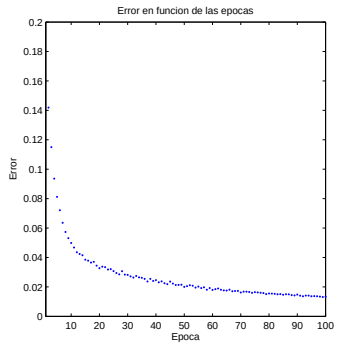
(b) Usando 20 neuronas.



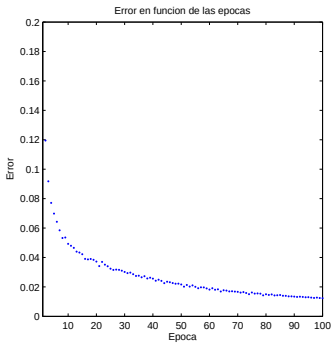
(c) Usando 25 neuronas.

Figura 3: Error durante el entrenamiento en el fold 1 del problema 1 usando una sola capa oculta y $\gamma = 0{,}1$.

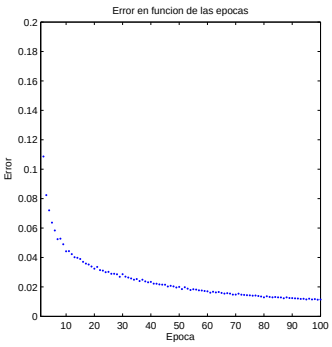
Considerando esos resultados, decidimos analizar qué sucedía al utilizar valores de γ mayores ya que tal vez era posible converger más rápidamente. Los resultados se encuentran en las Figuras 4, 5, 6 y 7, nuevamente para 15, 20 y 25 neuronas.



(a) Usando 15 neuronas.

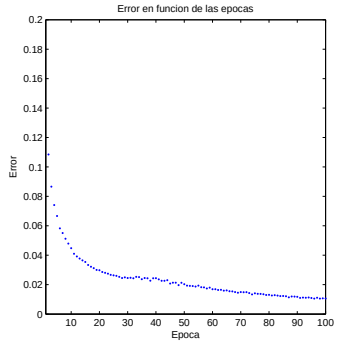


(b) Usando 20 neuronas.

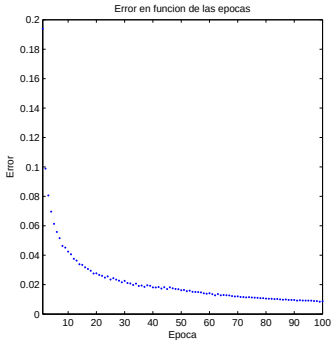


(c) Usando 25 neuronas.

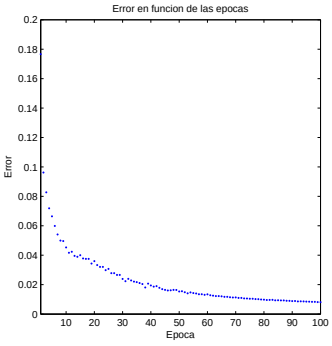
Figura 4: Error durante el entrenamiento en el fold 1 del problema 1 usando una sola capa oculta y $\gamma = 0{,}2$.



(a) Usando 15 neuronas.



(b) Usando 20 neuronas.

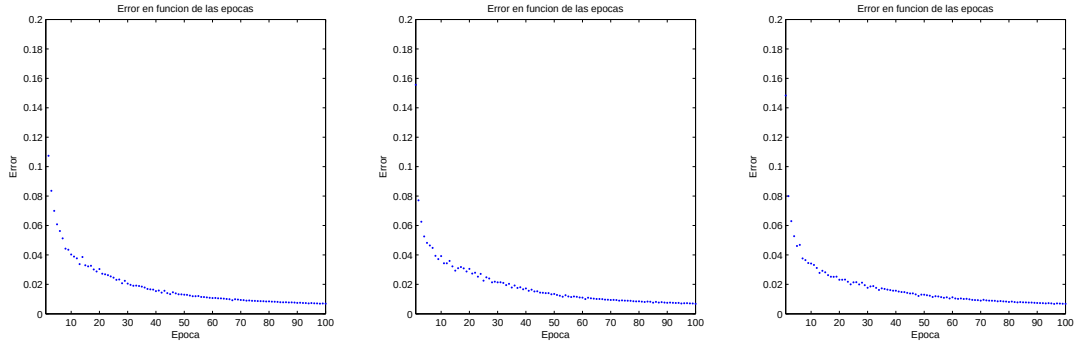


(c) Usando 25 neuronas.

Figura 5: Error durante el entrenamiento en el fold 1 del problema 1 usando una sola capa oculta y $\gamma = 0{,}3$.

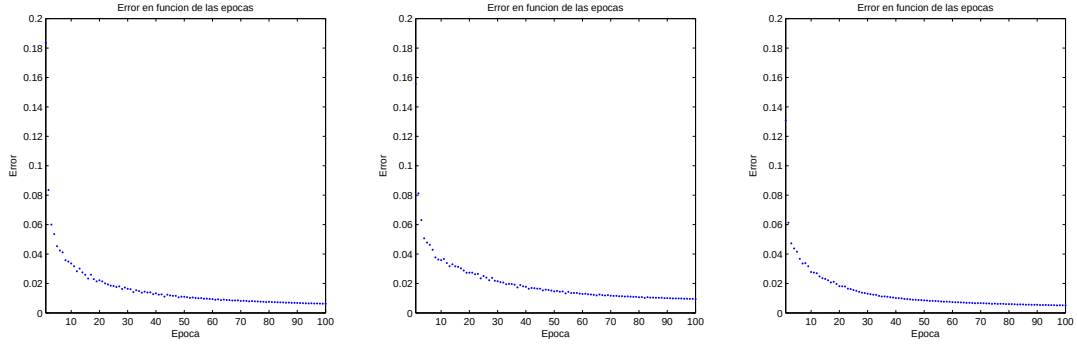
	15 neuronas	20 neuronas	25 neuronas
$\gamma = 0{,}1$	0.041432	0.043057	0.034287
$\gamma = 0{,}2$	0.039383	0.028796	0.045242
$\gamma = 0{,}3$	0.034535	0.033598	0.02312
$\gamma = 0{,}4$	0.034545	0.037957	0.036029
$\gamma = 0{,}5$	0.038391	0.03332	0.025975

Cuadro 1: Error de validación luego de 100 iteraciones en el fold 1 del problema 1.



(a) Usando 15 neuronas. (b) Usando 20 neuronas. (c) Usando 25 neuronas.

Figura 6: Error durante el entrenamiento en el fold 1 del problema 1 usando una sola capa oculta y $\gamma = 0{,}4$.

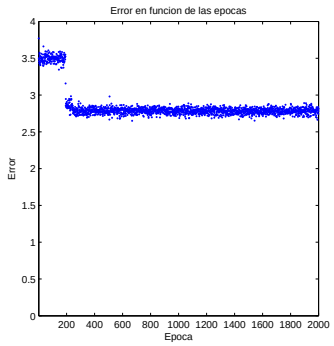


(a) Usando 15 neuronas. (b) Usando 20 neuronas. (c) Usando 25 neuronas.

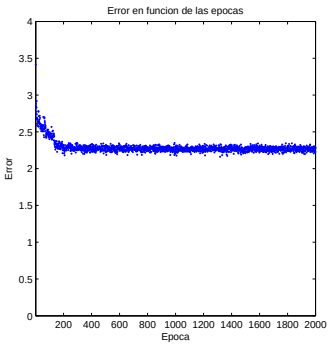
Figura 7: Error durante el entrenamiento en el fold 1 del problema 1 usando una sola capa oculta y $\gamma = 0{,}5$.

3.2. Problema 2

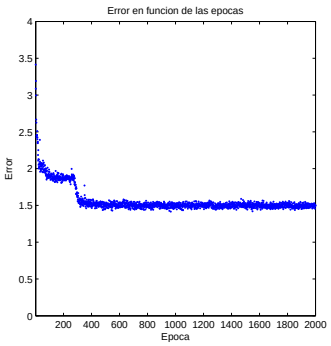
A continuación (Figura 8) mostramos los errores de entrenamiento en función de la época al entrenar las redes con diversas cantidades de neuronas en el fold 2 aunque comportamientos similares fueron observados en el resto de los folds. Se puede observar cómo la cantidad de neuronas en la capa oculta juega un papel esencial en el error obtenido. Dado que cuanto mayor es el número de neuronas, mejores resultados obtenemos, decidimos analizar los resultados usando 16, 17, 18, 19 y 20 neuronas, mostrados en la Figura 9.



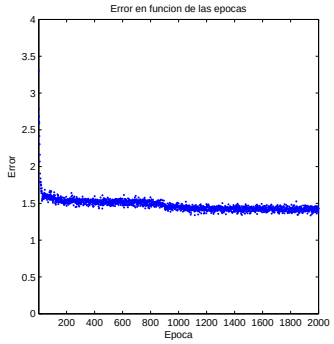
(a) Usando 2 neuronas.



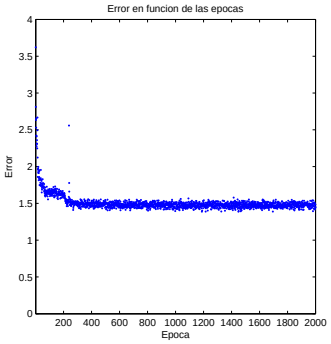
(b) Usando 4 neuronas.



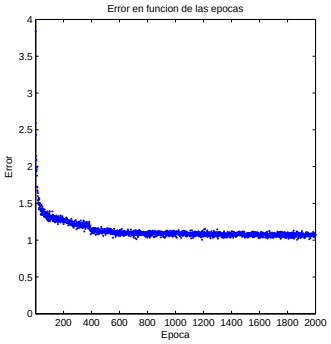
(c) Usando 6 neuronas.



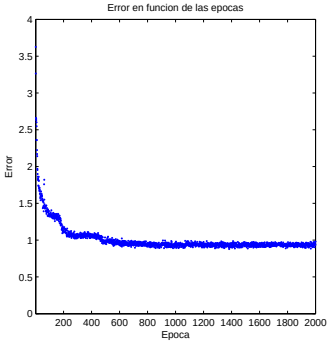
(d) Usando 8 neuronas.



(e) Usando 10 neuronas.

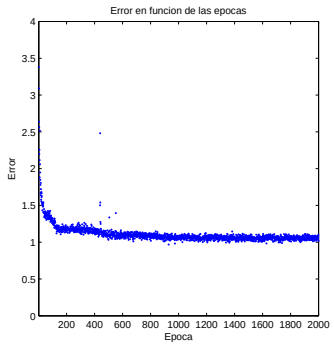


(f) Usando 12 neuronas.

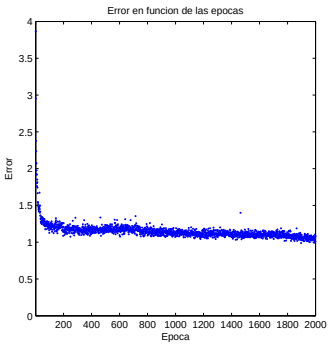


(g) Usando 14 neuronas.

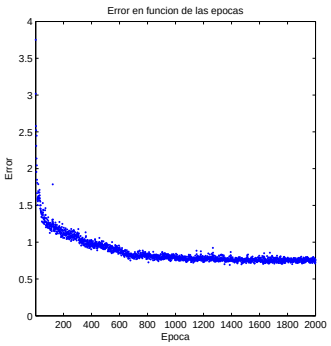
Figura 8: Error durante el entrenamiento en el fold 2 del problema 2 usando una sola capa oculta y $\gamma = 0{,}1$.



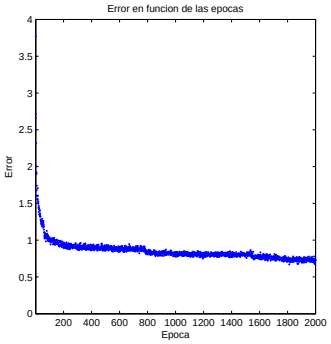
(a) Usando 16 neuronas.



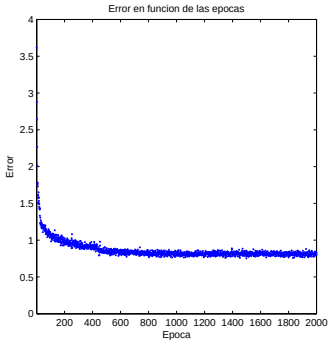
(b) Usando 17 neuronas.



(c) Usando 18 neuronas.



(d) Usando 19 neuronas.

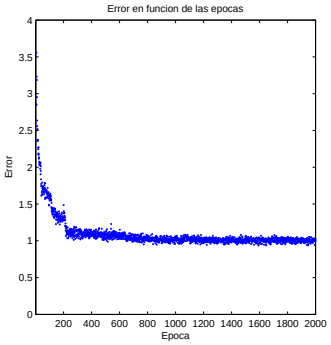


(e) Usando 20 neuronas.

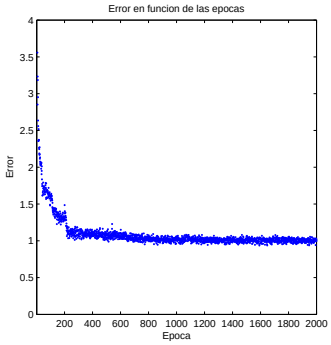
Figura 9: Error durante el entrenamiento en el fold 2 del problema 2 usando una sola capa oculta y $\gamma = 0{,}1$ segunda parte. Error en el conjunto de test (promedio entre los cuatro folds): 1,758475, 1,926875, 1,82305, 1,4711275 y 1,38239 respectivamente.

Tomando los resultados de las Figuras 8 y 9 se puede ver que los errores (tanto en entrenamiento como en validación) más bajos se obtienen usando 20 neuronas, además de un comportamiento estable por lo que decidimos no buscar configuraciones con más neuronas.

Considerando esa arquitectura, decidimos analizar otros valores de learning rate a fin de observar si es posible converger más rápido. Los resultados se encuentran en la Figura 10.



(a) Con $\gamma = 0{,}05$.



(b) Con $\gamma = 0{,}15$.

Figura 10: Error durante el entrenamiento en el fold 2 del problema 2 usando una sola capa oculta de 20 neuronas variando el valor de γ.

4. Conclusiones

5. Opciones de uso

Referencias

[1] John A. Hertz, Anders S. Krogh, Richard G. Palmer, *Introduction To The Theory Of Neural Computation*, Westview Press, June 24, 1991.

[2] Simon O. Haykin, *Neural Networks and Learning Machines*, Prentice Hall, 3rd Edition, November 28, 2008.

[3] Jeff T. Heaton, *Introduction to Neural Networks for Java*, Heaton Research, Inc. November 1, 2005.