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

Índice

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 en 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 análisis más robusto respecto al rendimiento. Esta técnica nos permitió tener una idea más real del nivel de generalidad de los resultados obtenidos con las redes.

2. Desarrollo

2.1. Implementación 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. Los pesos de las matrices son inicializados tomándolos de una distribución normal.

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..|w|] do
        y_{i+1} =activation([y_i 1] * w_i)
    end for
    return y
end function

function train(w, data, err_min, ep_max, γ, α)
    Δlw = ∅
    while e > err_min ∧ t < ep_max do
        e = 0
        t = t + 1
        for x, z ∈ data do
            y = propagateFeed(x)
            [fe Δw] = correction(w, y, z, γ)
            w = adaptation(w, Δw, Δlw, α)
            Δlw = Δw
            e = e + fe
        end for
    end while
end function
```

```
function correction(w, y, z, γ)
    Δw = ∅
    re = z − y_{|y|}
    fe = ||re||_1/|re|
    for i = [|w|.,1] do
        re = re .* activation'(y_{i+1})
        Δw_i = γ * [y_i 1]' * re
        re = re * w_{[1..|w|−1]}
    end for
    return [fe Δw]
end function

function adaptation(w, Δw, Δlw, α)
    for i = [1..|w|] do
        w_i = w_i + Δw + α * Δlw
    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 los mejores 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 la mayoria de los problemas puede ser resuelto con una única capa oculta[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 resultado

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 máximos de las mismas fueron elegidos a partir de que observamos que la red convergía 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 escogidas 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 en el problema 1, 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

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

2.4.6. Sigmoide bipolar vs. Sigmoide binario

No incluimos pruebas con el sigmoide bipolar en la experimentación ya que en una primera experimentación preliminar la red neuronal con el sigmoide binario se comportó mucho mejor en ambos ejercicios que con el sigmoide bipolar. Para ser más precisos, hubo una reducción del error del %50 utilizando el sigmoide binario bajo los mismos párametros.

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. Luego, variamos el learning rate (γ) y el momentum para el caso del problema 1. En el problema 2 no realizamos análisis del momentum, ya que los resultados obtenidos en el problema 1 respecto al mismo nos parecieron suficientes.

A continuación presentamos resultados sobre dichos experimentos.

3.1. Problema 1

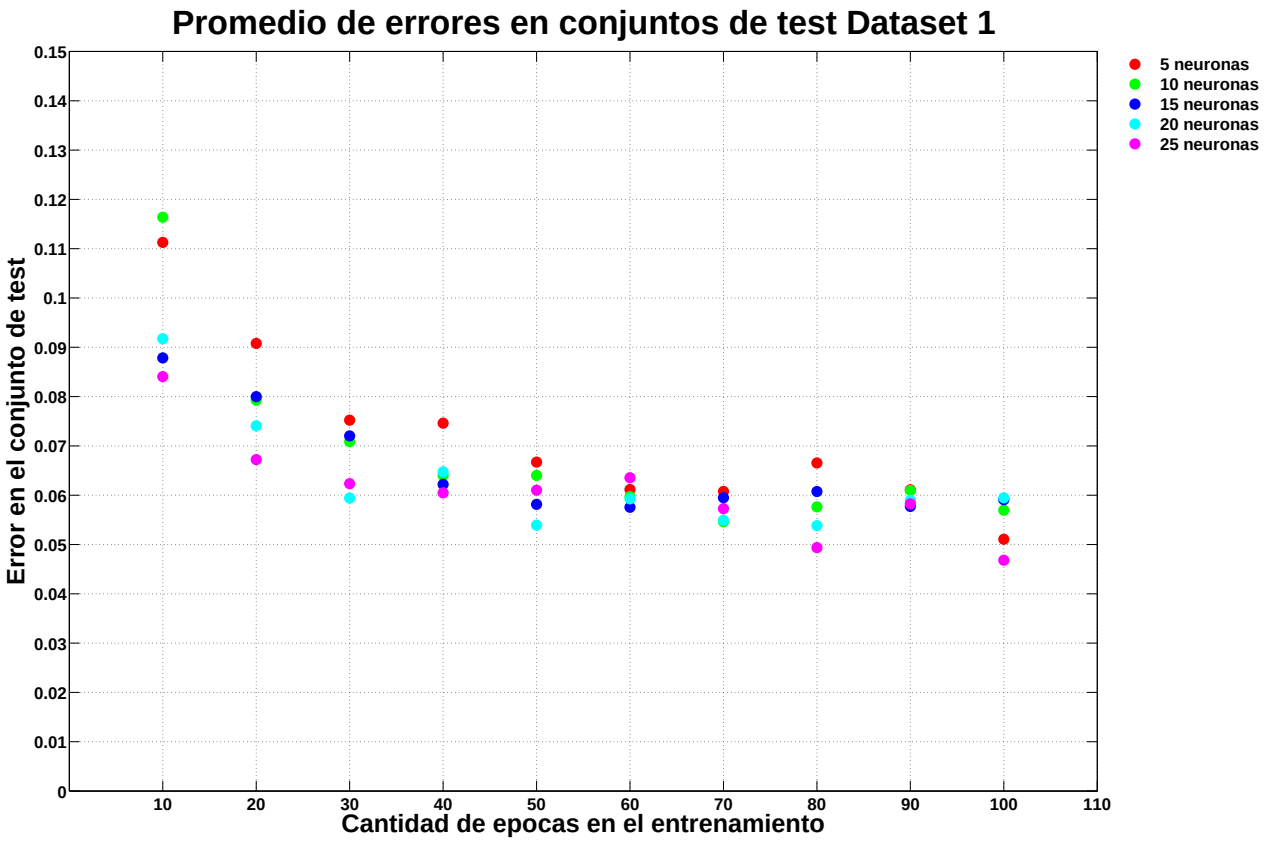
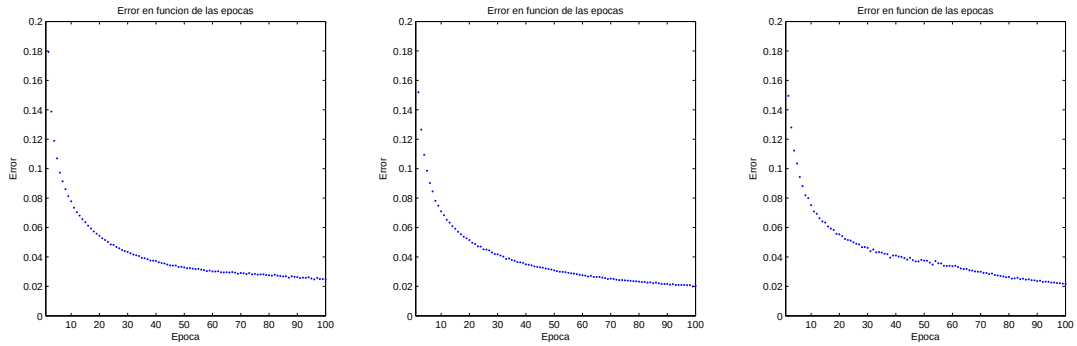


Figura 1: Promedio de error entre los 4 folds para el dataset 1 calculado sobre el conjunto de test usando $\gamma = 0{,}1$

3.1.1. Análisis de learning rate (γ)

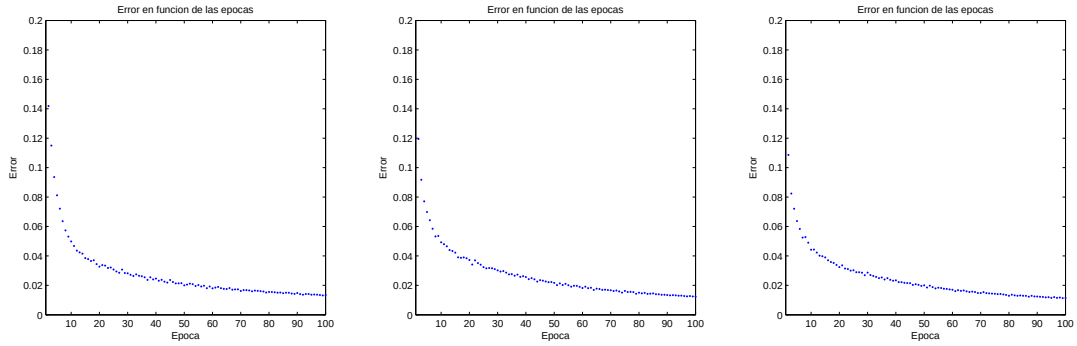
Dado que según la Figura 1 las configuraciones con 5 y 10 neuronas no presentan tan buenos resultados, mostramos a continuación (Figura 7) gráficos sobre la convergencia del error en función de las épocas usando las otras arquitecturas evaluadas. Para ello mostramos el fold 1 pero los demás presentan un comportamiento similar.



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

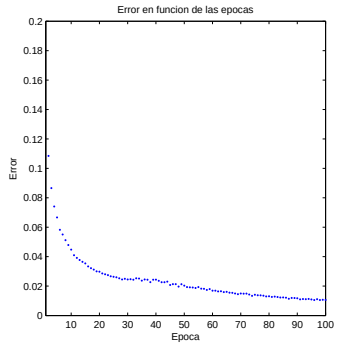
Figura 2: Error durante el entrenamiento en el fold 1 del problema 1 usando una sola capa oculta y $\gamma = 0{,}1$.

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 3, 4, 5 y 6, nuevamente para 15, 20 y 25 neuronas.

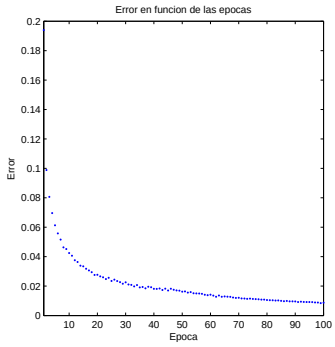


(a) Usando 15 neuronas. (b) Usando 20 neuronas. (c) Usando 25 neuronas.

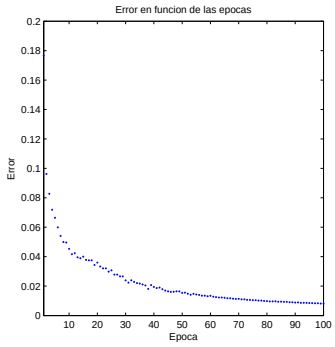
Figura 3: Error durante el entrenamiento en el fold 1 del problema 1 usando una sola capa oculta y $\gamma = 0{,}2$.



(a) Usando 15 neuronas.

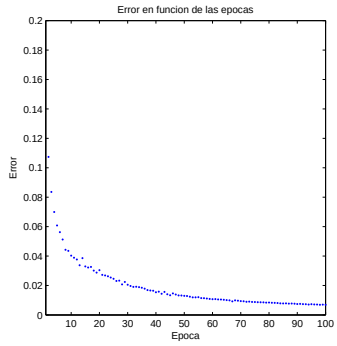


(b) Usando 20 neuronas.

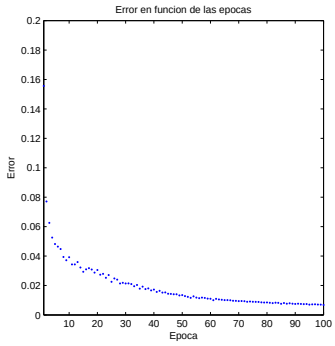


(c) Usando 25 neuronas.

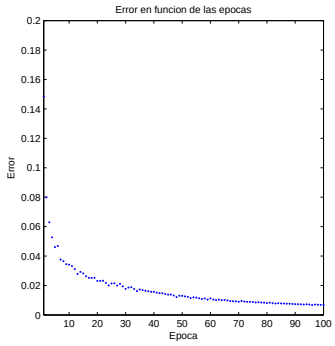
Figura 4: Error durante el entrenamiento en el fold 1 del problema 1 usando una sola capa oculta y $\gamma = 0{,}3$.



(a) Usando 15 neuronas.

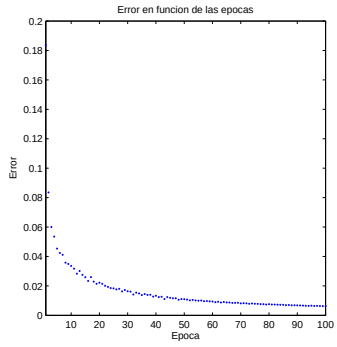


(b) Usando 20 neuronas.

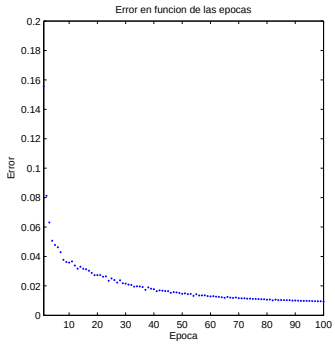


(c) Usando 25 neuronas.

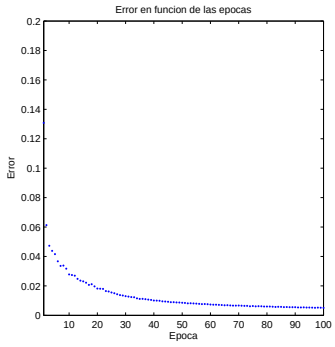
Figura 5: Error durante el entrenamiento en el fold 1 del problema 1 usando una sola capa oculta y $\gamma = 0{,}4$.



(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 $\gamma = 0{,}5$.

	15 neuronas	20 neuronas	25 neuronas
$\gamma = 0{,}1$	0.0590	0.0594	0.0468
$\gamma = 0{,}2$	0.0502	0.0469	0.0554
$\gamma = 0{,}3$	0.0485	0.0497	0.0448
$\gamma = 0{,}4$	0.0443	0.0521	0.0468
$\gamma = 0{,}5$	0.0447	0.0465	0.0481

Cuadro 1: Error de validación promedio luego de 100 iteraciones del problema 1.

En general, se puede observar una convergencia más rpida a medida que aumentamos el learning rate. Sin embargo, no debemos olvidar que a medida que aumentamos el mismo también incrementa el riesgo de que la red no converja.

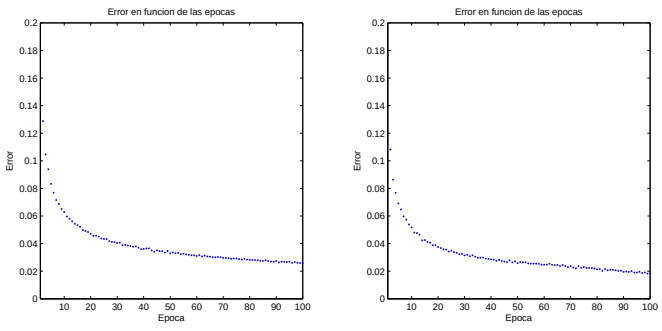
3.1.2. Análisis de momentum (α)

Para analizar el momentum decidimos utilizar la arquitectura de 20 neuronas, ya que tiene una buena convergencia y conserva más generalización que la arquitectura de 25 neuronas. El learning rate fue fijado en 0.1 ($\gamma = 0{,}1$) y los resultados son los promedios de los resultados de los 4 folds.

α	Error	Diferencia	α	Error	Diferencia
0.00	0.0594	0.0000	0.50	0.0564	-0.0030
0.05	0.0524	-0.0069	0.55	0.0520	-0.0074
0.10	0.0550	-0.0043	0.60	0.0544	-0.0050
0.15	0.0542	-0.0051	0.65	0.0511	-0.0082
0.20	0.0602	0.0007	0.70	0.0505	-0.0089
0.25	0.0543	-0.0051	0.75	0.0537	-0.0056
0.30	0.0534	-0.0059	0.80	0.0574	-0.0019
0.35	0.0565	-0.0029	0.85	0.0602	0.0007
0.40	0.0590	-0.0004	0.90	0.0533	-0.0061
0.45	0.0628	0.0034	0.95	0.0565	-0.0028

Cuadro 2: Error de validación promedio con 100 iteraciones y una capa oculta de 20 neuronas.

Como se puede observar, con casi todos los valores de momentum se observa una mejoría significativa respecto a la corrida sin el mismo, ocrriendo los máximos entre $\alpha = [0{,}55\ 0{,}70]$.



(a) $\alpha = 0$ (b) $\alpha = 0{,}70$

Figura 7: Error durante el entrenamiento sin momentum y con el valor de momentum fijado en $0,70$. Se puede observar una convergencia más rápida en el segundo caso.

3.2. Problema 2

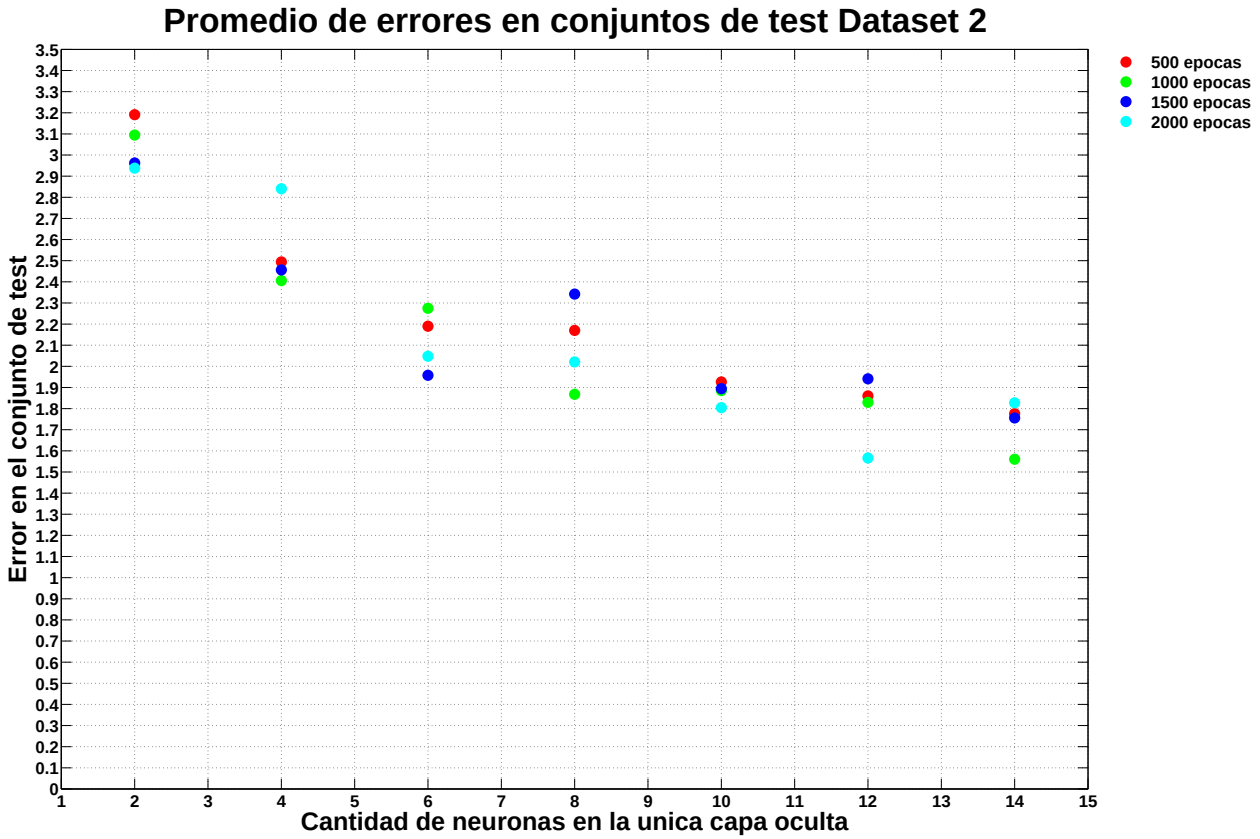
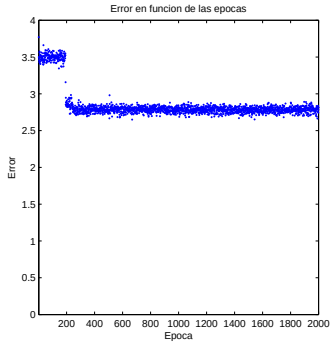
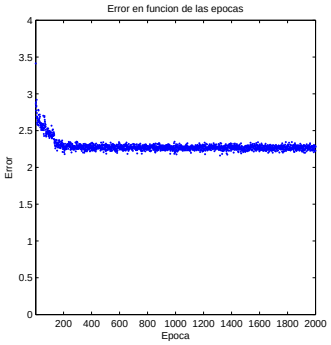


Figura 8: Promedio de error entre los 4 folds para el dataset 2 calculado sobre el conjunto de test usando $\gamma = 0{,}1$

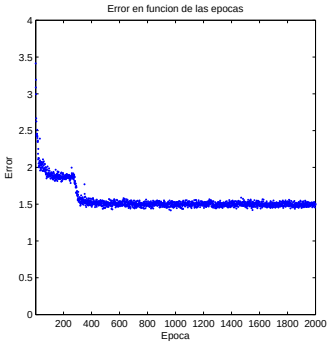
A continuación (Figura 9) mostramos los errores de entrenamiento en función de la época al entrenar las redes con diversas cantidades de neuronas en el fold 2 aunque comportamientos similares fueron observados en el resto de los folds. Se puede observar cómo la cantidad de neuronas en la capa oculta 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 10.



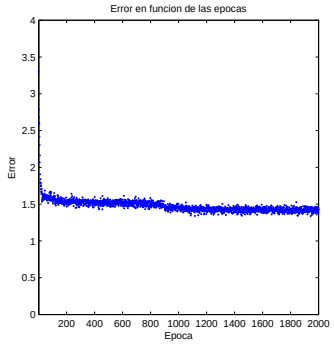
(a) Usando 2 neuronas.



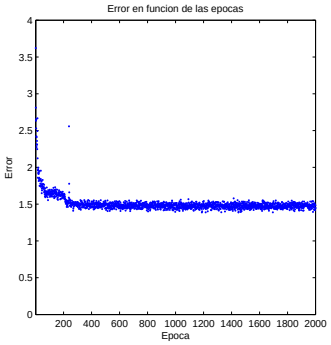
(b) Usando 4 neuronas.



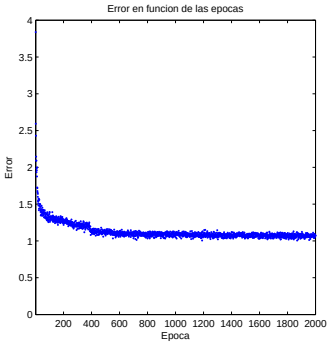
(c) Usando 6 neuronas.



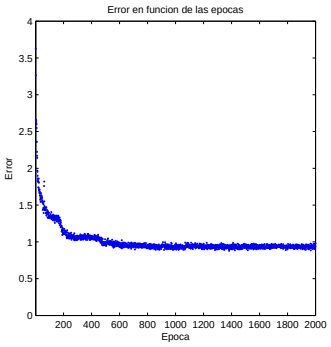
(d) Usando 8 neuronas.



(e) Usando 10 neuronas.

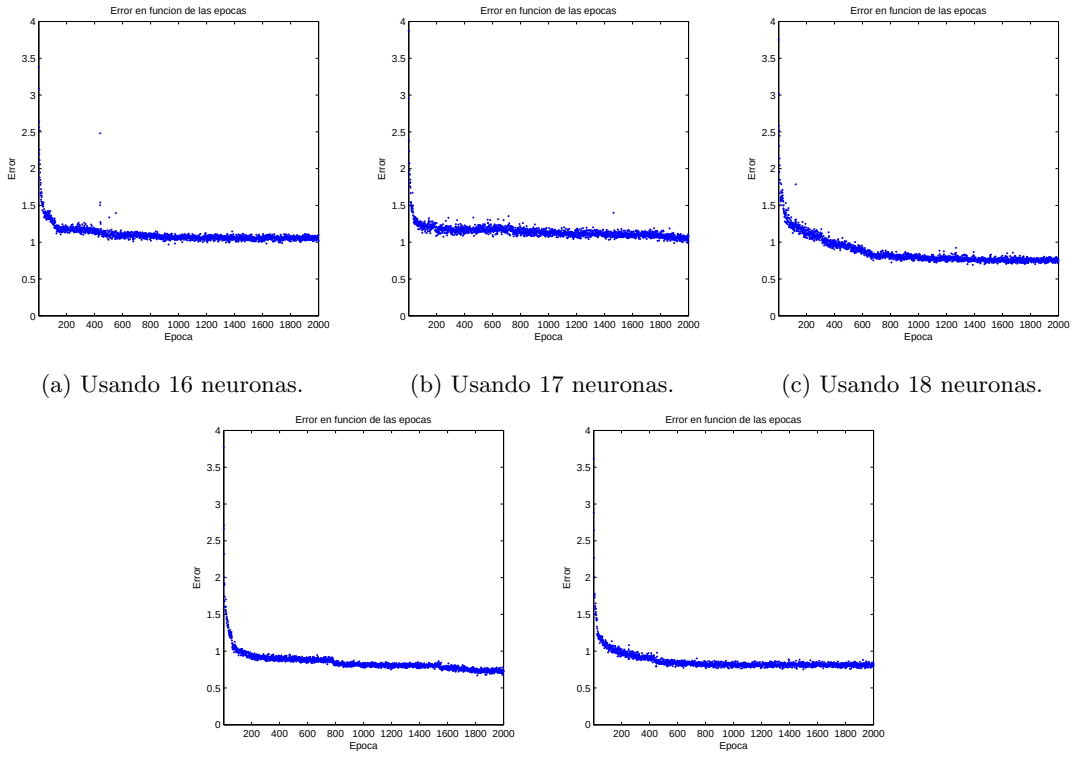


(f) Usando 12 neuronas.



(g) Usando 14 neuronas.

Figura 9: Error durante el entrenamiento en el fold 2 del problema 2 usando una sola capa oculta y $\gamma = 0{,}1$.



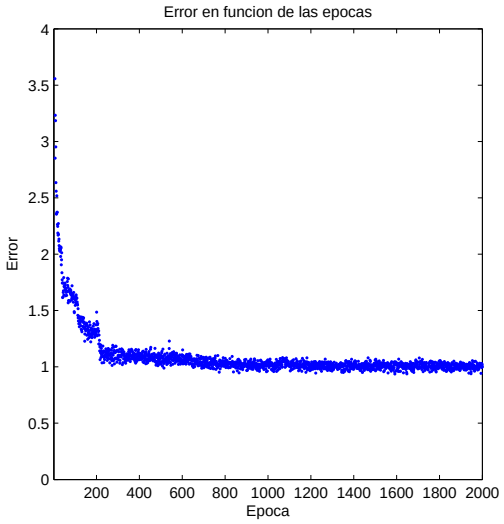
(a) Usando 16 neuronas. (b) Usando 17 neuronas. (c) Usando 18 neuronas.

(d) Usando 19 neuronas. (e) Usando 20 neuronas.

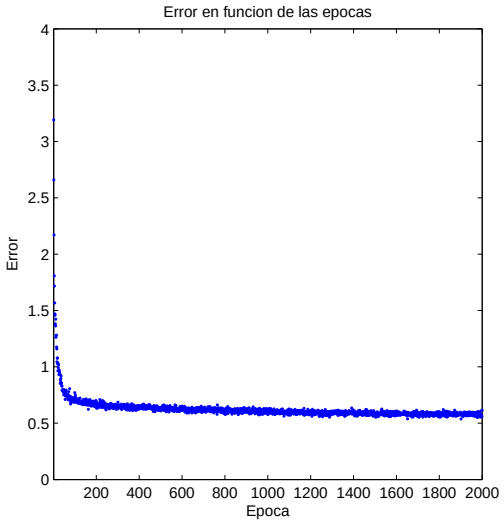
Figura 10: Error durante el entrenamiento en el fold 2 del problema 2 usando una sola capa oculta y $\gamma = 0{,}1$ segunda parte. Error en el conjunto de test (promedio entre los cuatro folds): 1,758475, 1,926875, 1,82305, 1,4711275 y 1,38239 respectivamente.

Tomando los resultados de las Figuras 9 y 10 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.

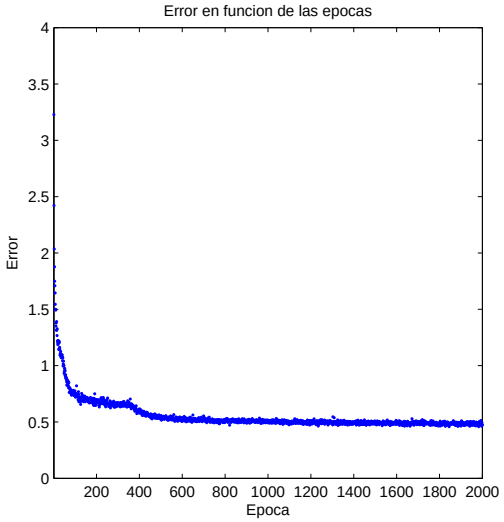
Considerando esa arquitectura, decidimos analizar otros valores de learning rate a fin de observar si es posible converger más rápido. Los resultados se encuentran en la Figura 11. Si bien es claro que cuanto menor es γ, más rápido obtenemos un error menor en este caso, si observamos los errores de validación, se puede ver cómo el mismo aumenta. Esto es un indicador de overfitting en el entrenamiento. En consecuencia, los mejores resultados son para $\gamma = 0{,}1$ ya que converge sin tanta variación como 0.15 pero mantiene un error de validación bajo.



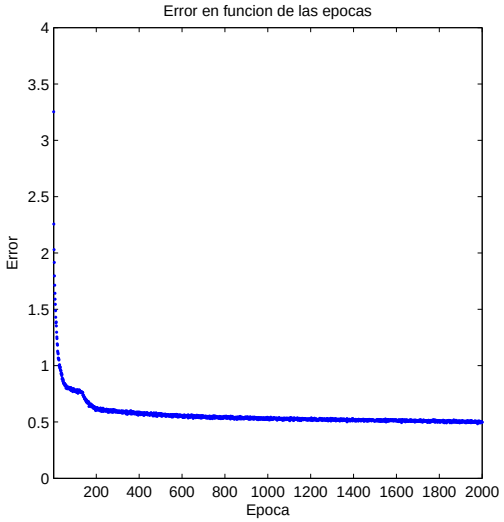
(a) Con $\gamma = 0{,}15$. Error en validación: 1.3657



(b) Con $\gamma = 0{,}05$. Error en validación: 1.5752



(c) Con $\gamma = 0{,}01$. Error en validación: 1.5528



(d) Con $\gamma = 0{,}005$. Error en validación: 1.6481

Figura 11: 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

4.1. Problema 1

Concluimos, observando los resultados, que una única capa oculta con 20 o 25 neuronas es suficiente para obtener una clasificación casi perfecta de los datos de test, dado un tiempo de entrenamiento adecuado.

Para este problema, parece ser que la red neuronal presenta un buen comportamiento con valores de learning rate y momentum altos, por lo que pueden aprovecharse para reducir el tiempo de entrenamiento o producir una precisión mayor en el mismo tiempo. Sin embargo, no consideramos adecuado incrementar demasiado estos valores, ya que la red puede terminar no convergiendo debido al incremento excesivo de los mismos en algún caso remoto.

Recomendamos entonces, una configuración de 20 neuronas en una única capa oculta y un entrenamiento con un valor de learning rate de 0.20, un valor de momentum de 0.70 y 1000 épocas.

4.2. Problema 2

A partir de los resultados analizados, es posible ver que un número de neuronas igual a 20 permite obtener bajo error en la validación. Podrían analizarse variantes con más neuronas o bien arquitecturas con menos neuronas pero ubicadas en más de una capa lo cual queda como trabajo futuro.

Si bien el error en la validación varía un poco según la cantidad de épocas utilizadas para el entrenamiento, valores mayores en general permiten obtener mejores resultados. Observando los gráficos correspondientes a la arquitectura con 20 neuronas se puede apreciar que con 500 épocas el error en el entrenamiento comienza a estabilizarse, sin embargo continúa disminuyendo hasta 1500 épocas, punto a partir del cual la disminución es cada vez menos apreciable.

Respecto del learning rate, por lo mencionado en la sección de resultados, el mejor valor entre aquellos considerados para la experimentación es 0.1 ya que proporciona un balance entre velocidad de convergencia y un bajo riesgo de que la red no converja.

4.3. Trabajo futuro

Dado el tiempo acotado para realizar pruebas con los conjuntos de datos, quedan planteados como posibles trabajos futuros el análisis de arquitecturas con más capas ocultas, distintos criterios de inicialización de pesos, cambios dinámicos en el learning rate y momentum dependientes del error observado, cambios en la función de error y distintos tipos de training (actualmente sólo probamos uno).

Si bien utilizar una sola capa en general debería dar buenos resultados, es posible que usando más de una capa sea posible converger a las soluciones obtenidas en la experimentación con una menor cantidad de iteraciones o bien utilizando menos neuronas en total y en menos tiempo.

Otra variante a evaluar es tratar de obtener un learning rate para cada problema que permita obtener buenos resultados más rápidamente. Si bien se evaluaron distintos valores, es posible continuar analizando algunos otros y combinar otras arquitecturas con otros learning rates.

Lo mismo podría plantearse en el problema 1 sobre el uso de momentum. En el problema 2, queda como trabajo a futuro evaluar si utilizar esta técnica permite una convergencia más rápida sin producir overfitting, aunque inferimos que un valor bajo del mismo ayudaría a la red en su convergencia.

Además, como observamos cierta dependencia en la convergencia de la red y su velocidad sobre la inicialización de los pesos, creemos que sería interesante investigar algún método que realice una cantidad de entrenamientos previos y cortos con el fin de quedarse con aquella inicialización que presente el menor error durante dichos entrenamientos.

5. Opciones de uso

El código principal para el funcionamiento de la red se encuentra en MyMultiPerceptron.m como fue explicado en el desarrollo del trabajo. Sin embargo, existen dos archivos que buscan facilitar la obtención de resultados de entrenamiento y validación de la red. A continuación detallamos las funciones en esos dos archivos.

5.1. Training

Realiza el entrenamiento de una red dado un conjunto de parámetros elegidos y genera un gráfico del error según la época.

```
function [ep_errors, final_error] = training(training_filename, hlayers, mode,
        output_filename, epochs, max_error, gamma, graph_filename, momentum)
```

- `training_filename` es el nombre del archivo a utilizar para entrenar. Por ejemplo, uno de los generados en las particiones.
- `hlayers` es el vector con las capas ocultas que se quiere en la red. Por ejemplo, `[2]` si se quieren dos neuronas en la capa oculta. Notemos que las cantidades de neuronas en las capas ocultas son independientes de las cantidades de entradas y salidas, las cuales son inferidas directamente de las dimensiones de los datasets.
- `mode` es alguna variante para indicar la elección entre binaria o bipolar o regresión. Sus opciones son: "bipolar", "binary", "bipolar-regresion", "binary-regresion". Los últimos dos difieren en los primeros utilizando una funcion lineal en la ultima capa. Este modo es necesario para el ejercicio 2.
- `output_filename` es el nombre del archivo donde se guardarán los parámetros de la red entrenada.
- `epochs` es el número máximo de épocas que se admite en el entrenamiento.
- `max_error` es el error máximo que se admite en el entrenamiento. Si el error de la red esta por debajo de este valor, el entrenamiento se detiene.
- `gamma` es el learning rate.
- `graph_filename` es el nombre del archivo donde se guardará el gráfico de error en función de las épocas.
- `momentum` es el valor del momentum. Este parámetro es opcional, de no ingresarse el valor se define en cero.
- `ep_errors` es el vector que guarda para cada época el error en esa etapa para el conjunto de entrenamiento mientras que `final_error` es el último valor del vector anterior.

5.2. Testing

Lee los datos sobre una red entrenada y calcula el error sobre un conjunto de datos de testing.

```
function error = testing(testing_filename, input_filename, gamma)
```

- `testing_filename` es el nombre del archivo sobre el cual se testeará la red.
- `input_filename` es el archivo con los parámetros de la red entrenada.
- `gamma` se utiliza para crear la instancia de la red.
- `error` es el cálculo del error sobre el conjunto de validación.

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.