Práctica 3: El Perceptrón Multicapa
Modelos de la Computación

Emilio Gomez Esteban

1. Enunciado de la prática

Descarga el fichero Multicapa.zip del campus virtual y súbelo a tu Matlab online. Para descomprimirlo, ejecuta en la línea de comandos de Matlab lo siguiente (unzip Multicapa.zip):

1. Los scripts 'PerceptronMulticapa.m' y 'PerceptronMulticapa_N_Pliegues.m' no se ejecutan bien porque es necesario implementar varias funciones. Ambos implementan un perceptrón multicapa con una única capa oculta con tantas neuronas como aparece en el primer elemento del vector Neu y la salida es una única neurona. 'PerceptronMulticapa.m' emplea un conjunto de entrenamiento, otro de validación y un último de test. 'PerceptronMulticapa_N_Pliegues.m' realiza una validación basada en N pliegues (con 10 pliegues).

   - Completa la implementación de las funciones 'logistica.m' y 'derivadaLogistica.m' que servirán para calcular la función de transferencia de cada una de las neuronas y para el método de retropropagación del error.
   - Completa la implementación de las funciones 'salidaRed.m' y 'retropropagacionError.m' que calculan respectivamente la salida de la red y los diferenciales de los pesos, esto último mediante el método de retropropagación del error visto en clase.
   - Sube al campus la implementación de cada una de estas cuatro funciones.

2. Utiliza el conjunto de datos 'Clasificacion/D Vertebral Column.mat' en el script 'PerceptronMulticapa.m'.

   - Copia una captura con la gráfica del MSE.
   - Atendiendo a los resultados, ¿en qué época crees que debería dejar de entrenar la red? ¿por qué?
   - Basándote en la época que has respondido en el apartado anterior, modifica el fichero 'PerceptronMulticapa.m' para que en la gráfica de resultados se pinte el punto con el MSE del conjunto de test de dicha época, copia una captura de dicha gráfica, ayúdate del siguiente código:

```
plot(indiceParada , MSEAcumuladoTest_vector(
   indiceParada), 'or')
```

- Comprueba qué ocurre cuando ejecutas el algoritmo con 2000 épocas, prueba en varias ejecuciones distintas para comprobar si el comportamiento varía de una ejecución a otra. Copia algunas capturas para ayudarte a explicar los resultados. ¿A qué crees que se debe la variabilidad de resultados?
- Pega en el documento de respuestas una gráfica con los valores objetivos de este conjunto de patrones, ¿cuál es el rango de la función a aprender?

(OPCIONAL) De aquí para abajo TODOS los apartados son opcionales

- ¿Qué crees que le estamos pidiendo a la red, que clasifique los patrones de entrada o que realice una regresión de una función general?
- Compara qué ocurre con el Error Cuadrático Medio (ECM) cuando usas el Beta actual 1.0 respecto de un valor de Beta de 0.1, ¿cuál es el ECM mínimo alcanzado en cada caso? (realiza varias simulaciones para tener un buen resultado medio del ECM) ¿a qué crees que se debe esta diferencia de ECM entre un beta de 1.0 y de 0.1? Pega capturas de las gráficas de resultados para cumplimentar tu respuesta.

3. Utiliza el conjunto de datos 'Regresion/D Parkinson Telemonit.mat' en el script 'PerceptronMulticapa_N_Pliegues.m':

   - Pega una gráfica con los valores objetivos de este conjunto de datos, ¿cuál es el rango de la función a aprender?
   - ¿Qué crees que le estamos pidiendo a la red, que clasifique los patrones de entrada o que realice una regresión de una función general?
   - Compara qué ocurre cuando ejecutas el algoritmo con 500 épocas utilizando una beta=1.0 respecto de una beta=0.1, ¿cuál es el ECM mínimo alcanzado en cada caso? ¿a qué crees que se debe esta diferencia? Pega capturas de las gráficas de resultados para cumplimentar tu respuesta.

En esta práctica hay que subir en total 4 script y un documento de texto con las respuestas, es decir:

- logistica.m
- derivadaLogistica.m
- retropropagacionError.m
- salidaRed.m
- Fichero de respuestas

2. Resolución de la práctica

Ejercicio 1

Apartado 1.1

Implementación de la función logistica.m:

```
function [Y] = logistica(X,beta)
%% Calcula la funcion logistica para cada uno de los
   elementos del vector columna X
    Y = 1./(1+exp(-2*beta*X));
end
```

Implementación de la función derivadaLogistica.m:

```
function [Y] = derivadaLogistica(X,beta)
%% Calcula la derivada de la funcion logistica para cada uno
   de los elementos del vector columna X
    Y = 2*beta*logistica(X,beta).*(1-logistica(X,beta));
end
```

Apartado 1.2

Implementación de la función salidaRed.m:

```
function [y, h, s, u] = salidaRed(patron, t, w, Beta)
%% Funcion que calcula la salida de la red (y), los pesos (h,
    s) y la salida de la capa oculta (s)
    u = t * patron';
    s = logistica(u,Beta);
    h = w * s;
    y=logistica(h,Beta);    %calculo de salida de la red,
       capa de salida
end

%{
    Estamos usando el modelo funcional para calcular la
       salida del modelo para un patron n-esimo x_n (ver
       formula de los apuntes de y gorro) usando la funcion
       logistica para g1 y g2 (funciones de transferencia).
%}
```

Implementación de la función retropropagacionError.m:

```
function [difW, difT] = retropropagacionError(patron, Z, y, w
   , s, h, u, Beta, eta)
%% Funcion que calcula los diferenciales de los pesos W y T

    %% Incializacion de variables
```

```
    nSalidas=size(y,1);
    nOcultas=size(w,2);

    delta2=zeros(nSalidas,1);
    difW=zeros(nSalidas, nOcultas);
    delta1=zeros(nOcultas,1);
    difT=zeros(nOcultas,size(patron,2));

    err = Z-y;
    g1d = derivadaLogistica(h,Beta);
    g2d = derivadaLogistica(u,Beta);

    %% --> Calculo de deltas2 y difW <--
    delta2 = err .* g1d;
    difW = eta * delta2 * s'; %size=nSalida,nOcultas

    %% --> Calculo de deltas1 y difT <--
    delta1 = (delta2 * w') .* g2d;
    difT = eta * delta1 * patron; %size=nOcultas,k

end
```

Apartado 1.3

✓

Ejercicio 2

Apartado 2.1

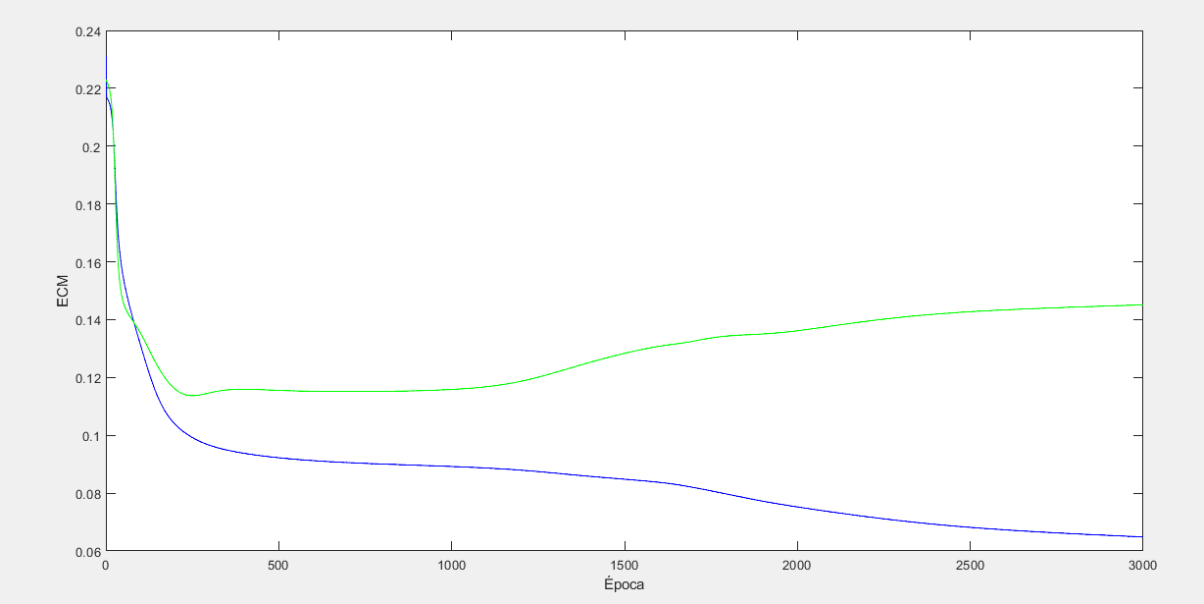

Figura 1: Gráfica del ECM

Apartado 2.2

Según la gráfica, parece que hay un claro punto de inflexión en la curva verde (el error de validación) donde el error comienza a aumentar nuevamente. Este comportamiento es un indicio de sobreajuste (overfitting): la red sigue mejorando en los datos de entrenamiento (curva azul), pero empieza a empeorar en los datos de validación (curva verde) después de cierto punto.

La red debería dejar de entrenarse justo antes de que el error de validación (curva verde) comience a aumentar. Un punto adecuado para detener el entrenamiento parece estar alrededor de la época 500, ya que antes de la época 500 el ECM de validación es más bajo, lo que significa que la red está generalizando mejor.

Deberíamos dejar de entrenar la red en ese punto para evitar el sobreajuste ya que el modelo continúa ajustándose a los datos de entrenamiento, pero está perdiendo su capacidad de generalizar en datos no vistos, como lo sugiere el aumento en el error de validación. La época donde el error de validación es mínimo refleja el mejor compromiso entre ajuste a los datos de entrenamiento y generalización a nuevos datos.

Apartado 2.3

Hemos añadido al final del código de PerceptronMulticapa.m las siguientes líneas:

```
plot(MSEAcumuladoTest_vector,'r')
hold on
[~,indiceParada] = min(MSEAcumuladoValid_vector);
plot(indiceParada , MSEAcumuladoTest_vector(indiceParada), '
   or')
legend('MSE Entrenamiento', 'MSE Validacion', 'MSE Test en
   mejor epoca')
```

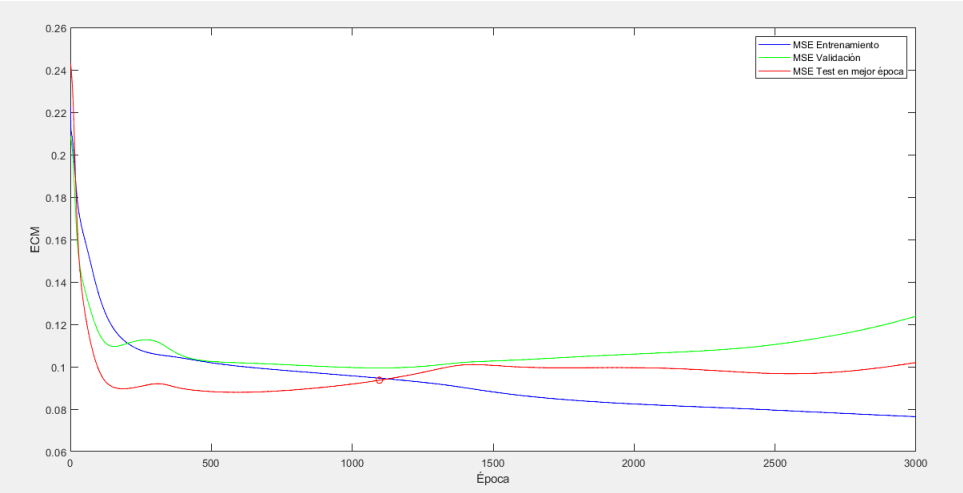

Figura 2: Gráfica del ECM con índice de parada

Apartado 2.4

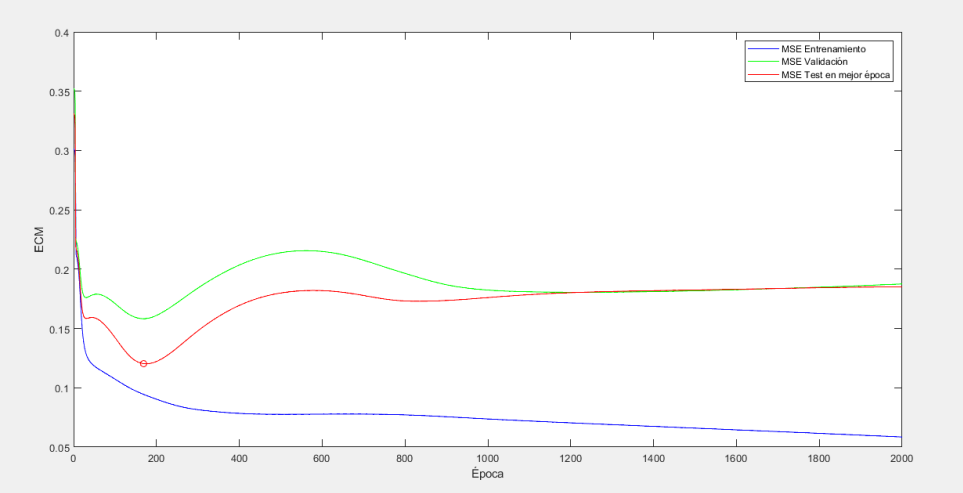

Figura 3: Gráfica del ECM (2000 épocas)

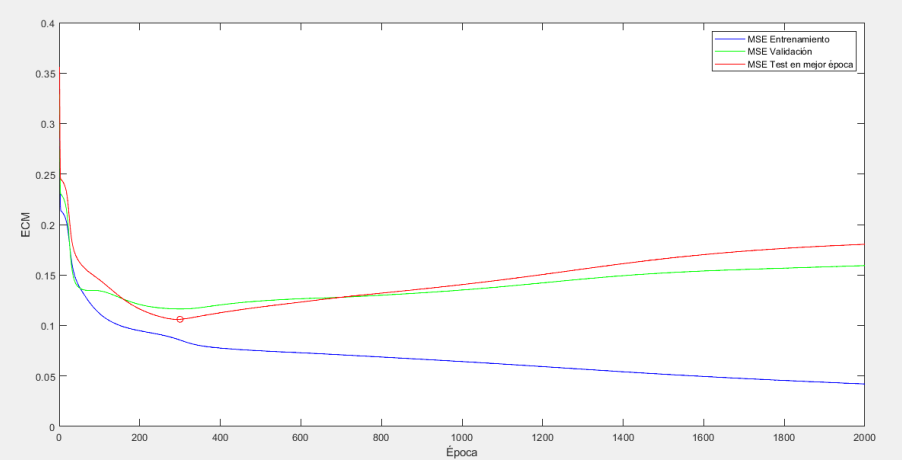

Figura 4: Gráfica del ECM (2000 épocas)

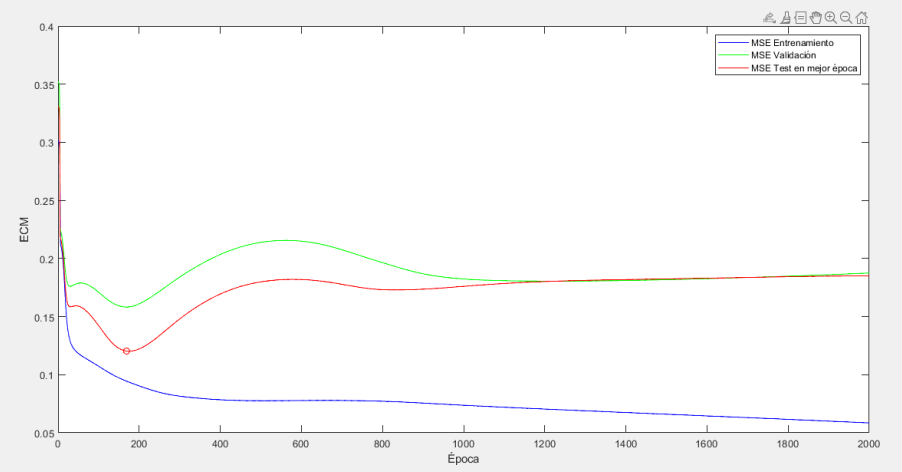

Figura 5: Gráfica del ECM (2000 épocas)

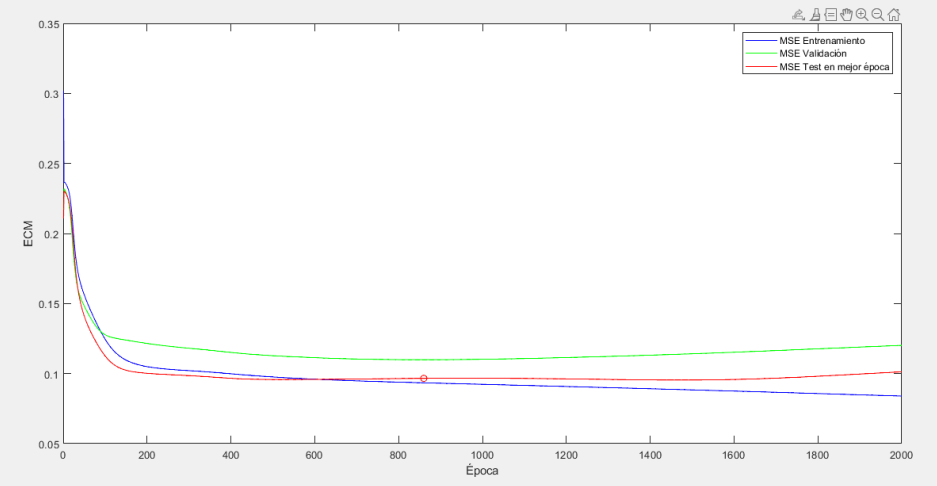

Figura 6: Gráfica del ECM (2000 épocas)

La variabilidad en los resultados de un algoritmo como el perceptrón multicapa en distintas ejecuciones se puede deber principalmente a varios factores, entre ellos: la inicialización aleatoria de los pesos (diferentes inicializaciones pueden hacer que el entrenamiento converja a distintos mínimos locales) y la elección de los algoritmos de optimización utilizados para la retropropagación (la presencia de múltiples mínimos locales en la función de costo puede llevar a diferentes resultados dependiendo de la ejecución).

Apartado 2.5

Hemos añadido las siguientes líneas de código para representar una gráfica con los valores objetivos de este conjunto de patrones:

```
figure;
plot(Z, 'or');  % Grafica de los valores objetivos
xlabel('Indice de los patrones');
ylabel('Valor objetivo (clase)');
title('Valores Objetivos de los Patrones');
```

Hemos obtenido la salida:

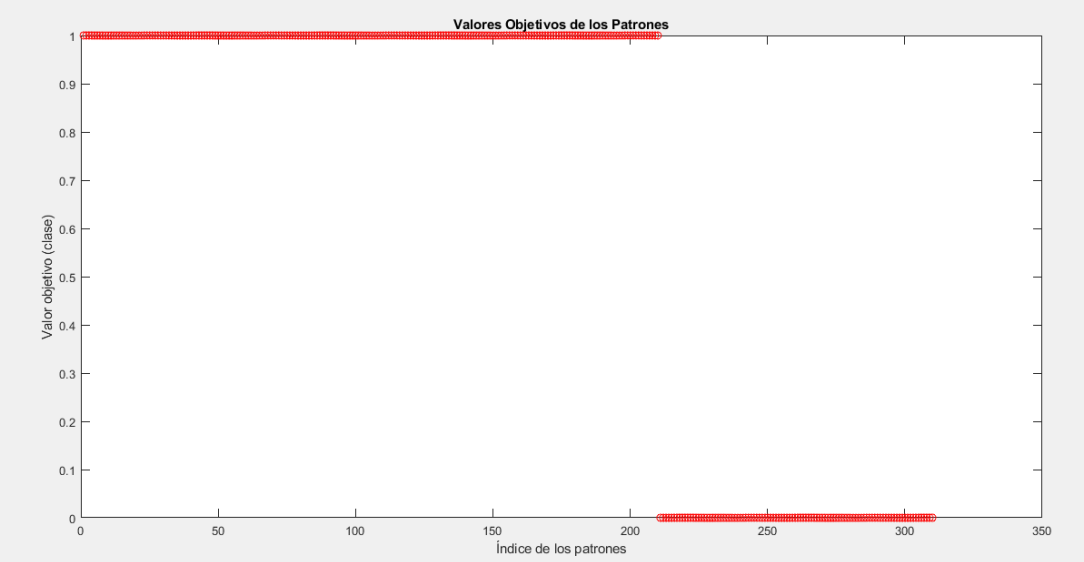

Figura 7: Gráfica con los valores objetivos del conjunto de patrones

Dado que el conjunto de datos 'Vertebral Column' es un problema de clasificación binaria, los valores objetivos (Z) de salida son clases binarias (en este caso $\{0, 1\}$), lo que implica que el rango de la función a aprender es discreto.

Apartado 2.6

Lo que le estamos pidiendo a la red es que clasifique los patrones de entrada. Esto es debido a que los valores objetivo son clases discretas. En los problemas de clasificación, la red intenta asignar cada patrón a una clase (discreta) y no aproximar un valor numérico continuo. El objetivo es predecir etiquetas discretas.

Según el código y el conjunto de datos, todo parece estar orientado a resolver un problema de clasificación (por ejemplo, las funciones de error y las configuraciones de la red neuronal). Además, la forma en que se distribuyen los datos de entrenamiento, validación y prueba, así como los cálculos de error acumulado, son típicos en tareas de clasificación.

Apartado 2.7

El parámetro Beta en las redes neuronales, especialmente en las funciones de activación como la sigmoide logística, controla la pendiente de la función en su región central. Un cambio en Beta afecta la forma en que las neuronas reaccionan a las entradas, lo que influye directamente en la tasa de aprendizaje y la capacidad de la red para ajustar los pesos durante el entrenamiento.

Las diferencias entre usar un Beta=1.0 y un Beta=0.1 es que con el segundo el ECM es más alto, ya que la función sigmoide se suaviza mucho más. Esto hace que las actualizaciones de pesos sean más pequeñas en cada paso, lo que puede hacer que la red tarde más en converger, pero también reduce el riesgo de sobreajuste y oscilaciones grandes. La red se vuelve menos sensible a los pequeños cambios en las entradas, lo que puede resultar en una convergencia más lenta pero más suave a diferencia de usar Beta=1.0.

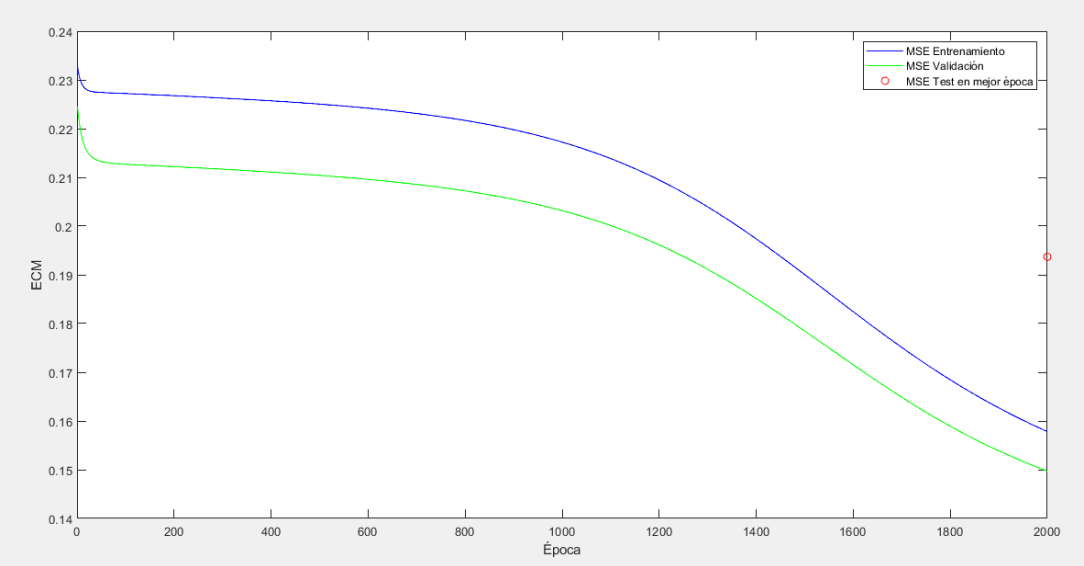

Figura 8: Gráfica del ECM con Beta=0.1

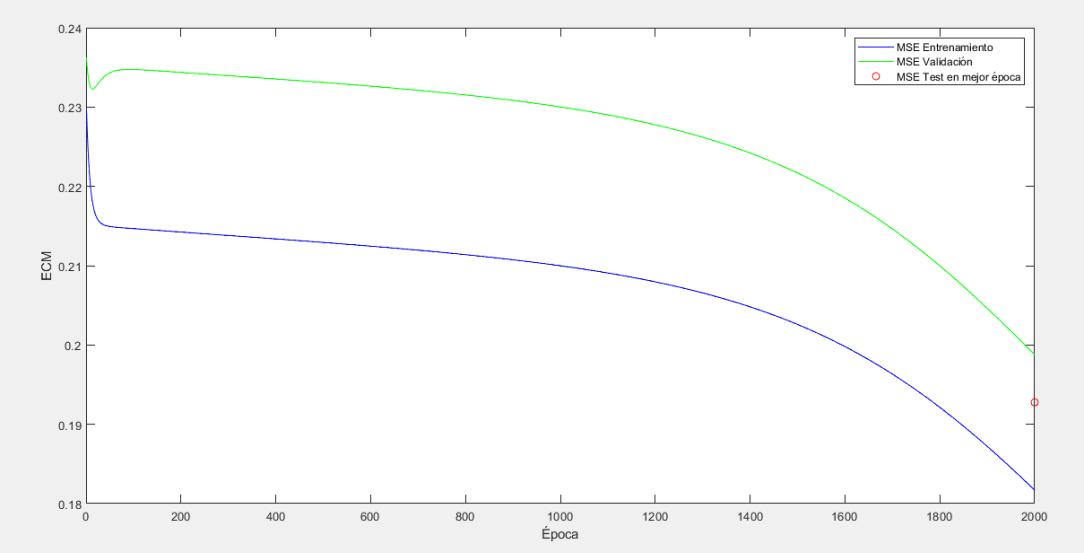

Figura 9: Gráfica del ECM con Beta=0.1

Ejercicio 3

Apartado 3.1

Hemos añadido las siguientes líneas de código para representar una gráfica con los valores objetivos de este conjunto de patrones:

```
figure;
plot(Z, 'or');  % Grafica de los valores objetivos
xlabel('Indice de los patrones');
ylabel('Valor objetivo (clase)');
title('Valores Objetivos de los Patrones');
```

Hemos obtenido la salida:

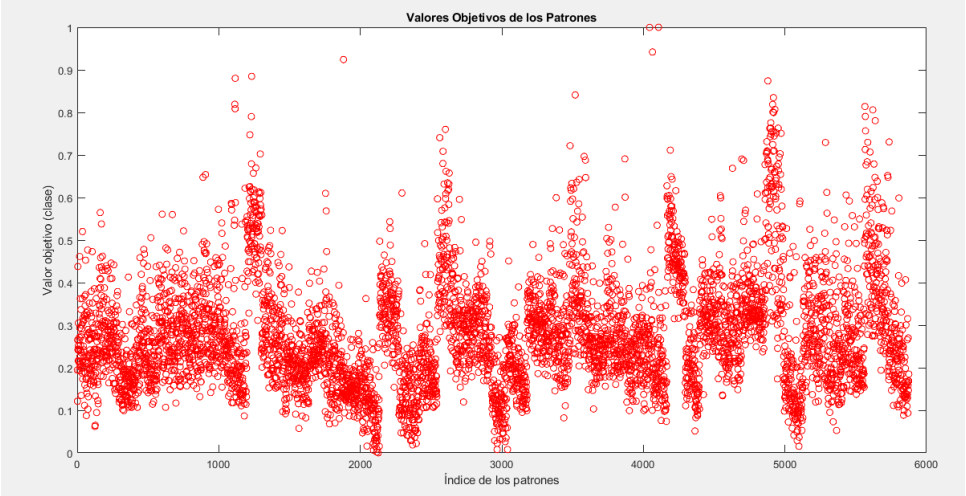

Figura 10: Gráfica con los valores objetivos del conjunto de patrones

Hemos obtenido que los valores objetivos (Z) de salida son continuos, en este caso el intervalo $[0, 1]$, lo que implica que el rango de la función a aprender es continuo (hemos puesto un máximo de 10 épocas porque sino no acababa el algoritmo).

Apartado 3.2

En este caso, estamos pidiendo a la red que realice una regresión en lugar de clasificar patrones de entrada, ya que estamos buscando aproximar una función que relacione los patrones de entrada con valores de salida continuos. . Además, la elección de los conjuntos de datos (como D_Parkinson_Telemonit.mat) también sugiere que se están utilizando para tareas de regresión, dado que muchos de esos conjuntos están diseñados para predecir un valor continuo (como la severidad de una enfermedad, por ejemplo) en lugar de clasificar ejemplos en categorías discretas.

Apartado 3.3

Beta es un parámetro que controla la tasa de activación de las neuronas en la red. Un valor alto (1.0) puede llevar a una respuesta más agresiva a los errores durante la retropropagación, lo que puede resultar en una convergencia más rápida pero también puede causar problemas de sobreajuste o oscilaciones en el proceso de entrenamiento. Un valor bajo (0.1) podría resultar en un aprendizaje más estable, pero más lento, lo que podría conducir a una convergencia más lenta y, potencialmente, a un mejor rendimiento general en datos no vistos, dependiendo del problema.