PARCIAL II

COMPUTACIÓN BLANDA

DANIEL DIAZ GIRALDO

DANIEL FELIPE MARIN

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

PEREIRA, RISARALDA

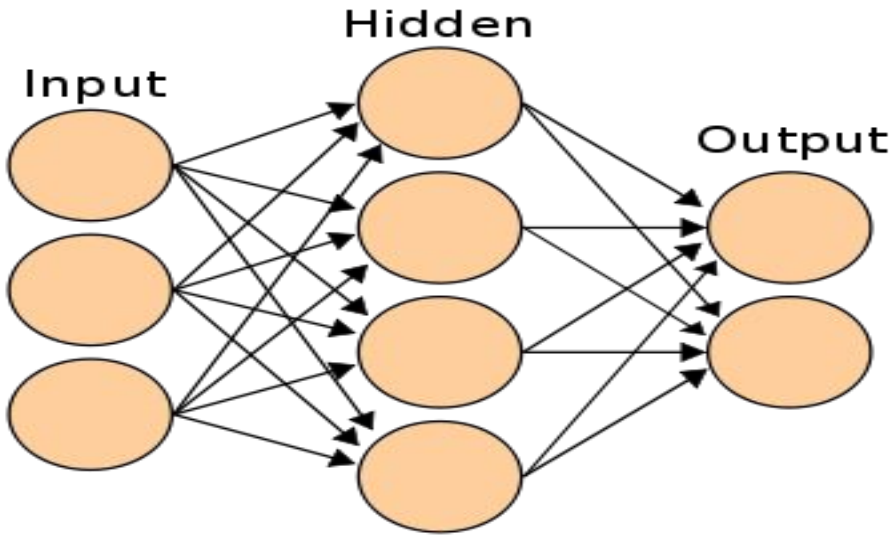
Metodología:

1. Carga de datos
    a. Carga de dataset en el entorno de octave, y su visualización.
2. Elaboración del modelo funcional:
    a. Adecuación de una red neuronal para la clasificación de 10 clases.
    b. Propagación hacia adelante.
    c. Propagación hacia atrás.
    d. Función de costo.
    e. Verificación del gradiente.
3. Toma de datos y evaluacion conceptual.
    a. Resultados.

Pasos realizados:

Algoritmo de multi-clasificación usando una red neuronal.

- Esquema utilizado:



Input layer: X[m x n] : Corresponde a nuestros datos de entrada, con la cual la red neuronal será entrenada para clasificar. m= 50.000 n=3072.

Hidden layer: Correspondiente a las funciones que harán la clasificación de la red, tamaño de 300 neuronas.

Output layer: Neuronas que contendrá el criterio de clasificación, gracias al proceso de las neuronas anteriores.Tamaño de 10 neuronas.

- Sustracción correcta del dataset de imágenes, el cual comprendía una colección de 60 mil imágenes comprendidas en 50 mil imágenes de entrenamiento y 10 mil imágenes de testing , todo este dataset comprendido entre 10 posibles categorías:
    - airplane
    - automobile
    - bird
    - car
    - deer
    - dog
    - frog
    - horse
    - ship
    - truck

Cada imagen tenía el tamaño de 32x32 conjunto a los 3 canales RGB, lo que generó una matriz del tamaño de 3072 para cada imagen. los labels están en formato correspondiente al orden de la lista (ejemplo: si es un caballo su valor correspondiente en el label es 8)

- Luego de tener los datos con los cuales trabajará la red, se procede a re-codificar los labels para tener un vector del tamaño igual a las neuronas de salida, esto es debido al proceso de clasificación binaria y al dominio de la función de activación (función sigmoidal).
- Se generan los pesos de manera random y con una tasa de cambio de epsilon, cuyo dominio generará valores positivos y negativos, esto con el fin de evitar el fenómeno de symmetry breaking
- Para refinar el aprendizaje de la red, se pasa una propagación hacia adelante y luego una hacia atrás, para cada iteración, a su vez se actualizan los valores de los pesos (W's).
- Al finalizar el proceso iterativo, pasamos una última propagación hacia adelante y se mide el factor de error para las muestras tomadas.

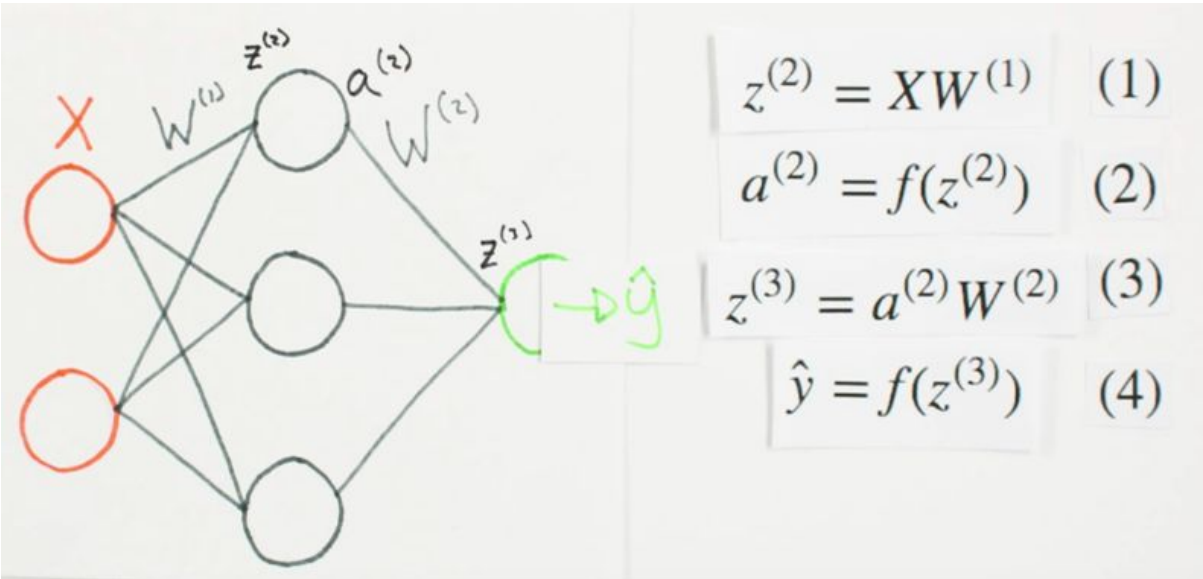
- Chequeo de gradiente:

Se guardan en un vector las derivadas W obtenidas en el backpropagation (grad).
Iterando se varía una posición, dada por la iteración, del vector gradiente que contiene las derivadas de W. Las variaciones consisten en una suma y resta de un epsilon del orden de 10e-^5, posterior a la variación se reconstruyen W0 y W1(diferentes para suma y resta del epsilon) para luego calcular la función de costo J con los Wi+E y Wi-E, y poder aplicar el concepto de derivada numérica.
Se calcula un vector que contiene el gradiente numérico usando concepto de la derivada numérica :

$$\frac{d}{d\theta}J(\theta) = \lim_{\epsilon \to 0} \frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon}.$$

Cada derivada numérica de una iteración se guarda en un vector llamado gradnum, para luego calcular la diferencia entre el gradnum y el grad, que debe dar en un orden de 10e-^8 o menor, se aplica la formula: **diff= norm(grad-gradnum)/norm(grad+gradnum).**
Si se cumple que la diferencia es menor o igual a 10e-^8, se puede asegurar que la red está funcionando correctamente.

- ForwardPropagation (Propagación hacia adelante):
  Propagación hacia adelante de la entrada (imagenes) para un patrón de entrenamiento (modelo) a través de la red neuronal con el fin de generar activaciones de salida de la propagación (output layer) . Determina que tan bueno es nuestro modelo referente a los datos de entrada, dando como resultado predicciones en la capa de salida.



$$z^{(2)} = XW^{(1)} \quad (1)$$
$$a^{(2)} = f(z^{(2)}) \quad (2)$$
$$z^{(3)} = a^{(2)}W^{(2)} \quad (3)$$
$$\hat{y} = f(z^{(3)}) \quad (4)$$

Pseudocódigo ForwardPropagation -
https://www.youtube.com/watch?v=UJwK6jAStmg

- BackPropagation (Propagación hacia atrás)
  El método calcula el gradiente de una función con respecto a todos los pesos en la red. El gradiente se alimenta al método de optimización que a su vez lo utiliza para actualizar los pesos , en un intento de minimizar la función de pérdida o costo .
  -Obtención de derivadas:

$$\frac{\partial J}{\partial W^{(2)}} = (a^{(2)})^T \delta^{(3)}$$

$$\delta^{(3)} = -(y - \hat{y}) f'(z^{(3)})$$

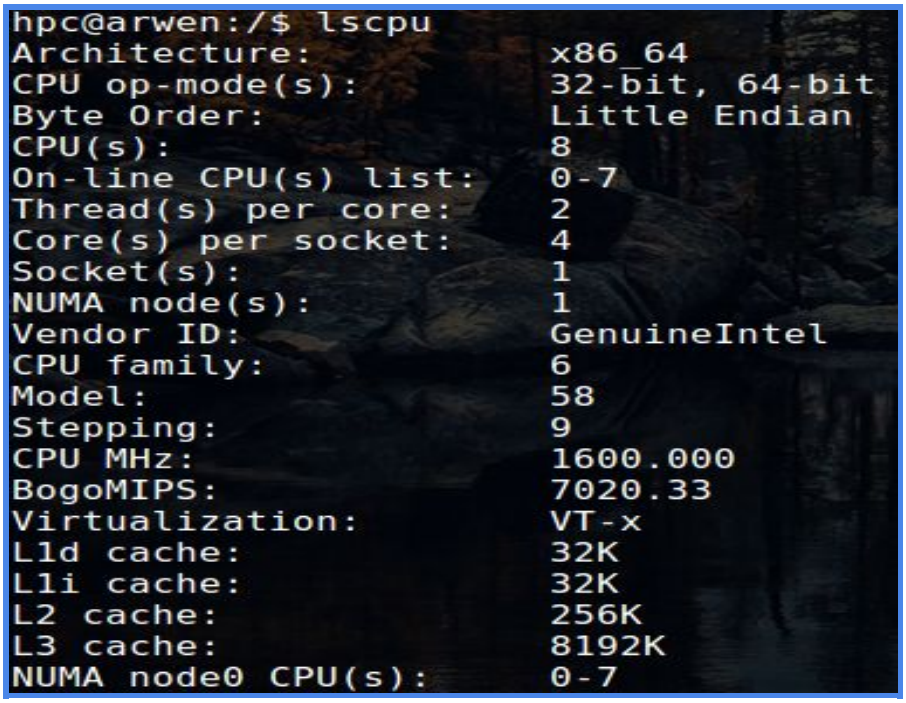
$$\frac{\partial J}{\partial W^{(1)}} = X^T \delta^{(2)}$$

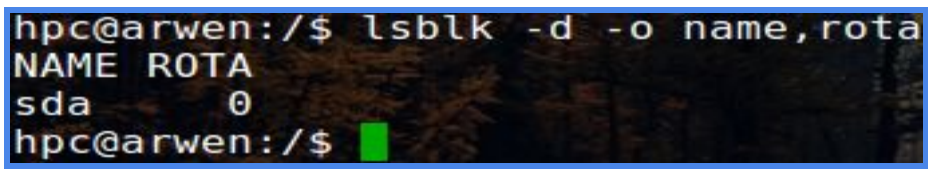
$$\delta^{(2)} = \delta^{(3)} (W^{(2)})^T f'(z^{(2)})$$

- Función de costo:
  - Función a ser minimizada, error cuadrático.

$$J = \sum \frac{1}{2} (y - \hat{y})^2 \quad (5)$$

Características del ordenador:





ssd

Resultados

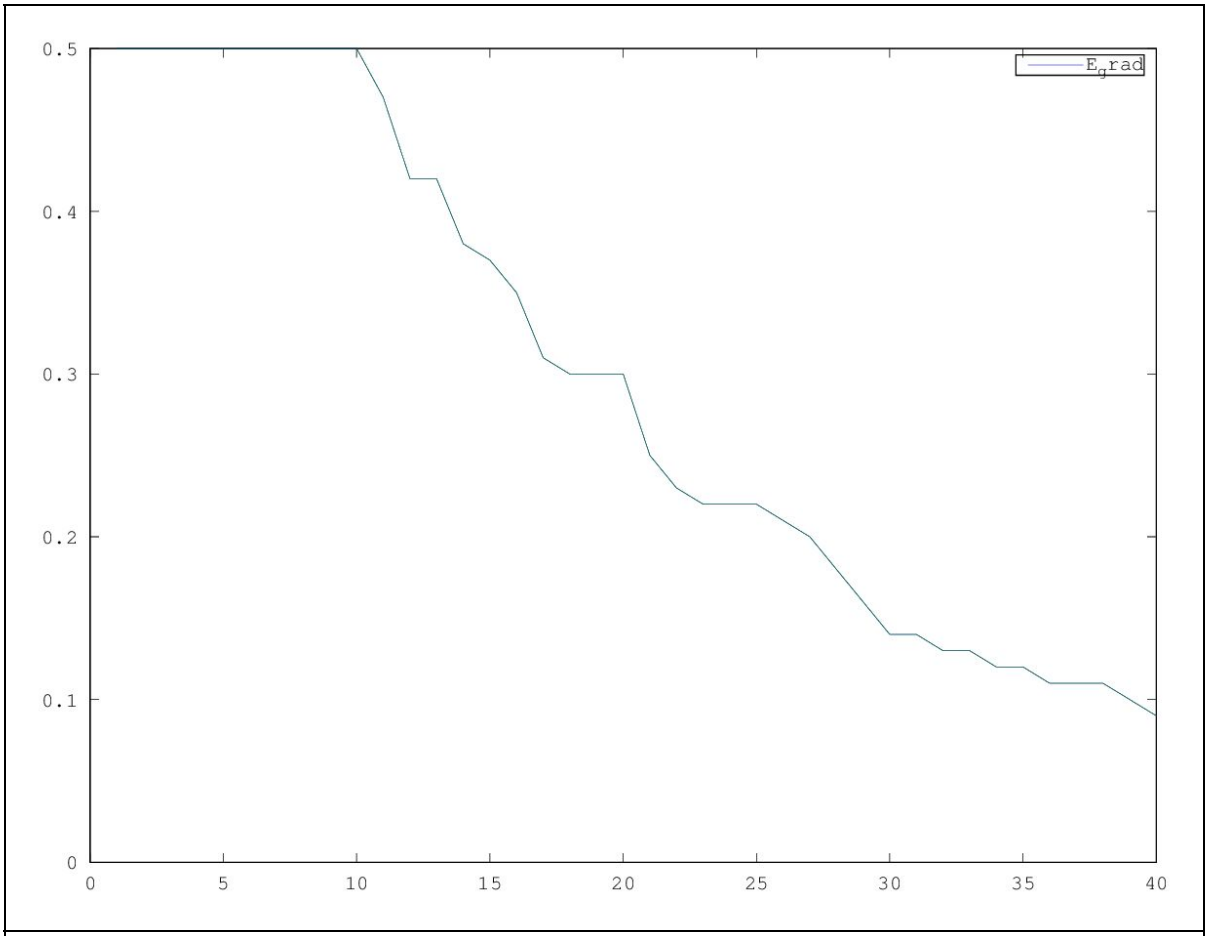
1.Verificación del gradiente(1 iteración).

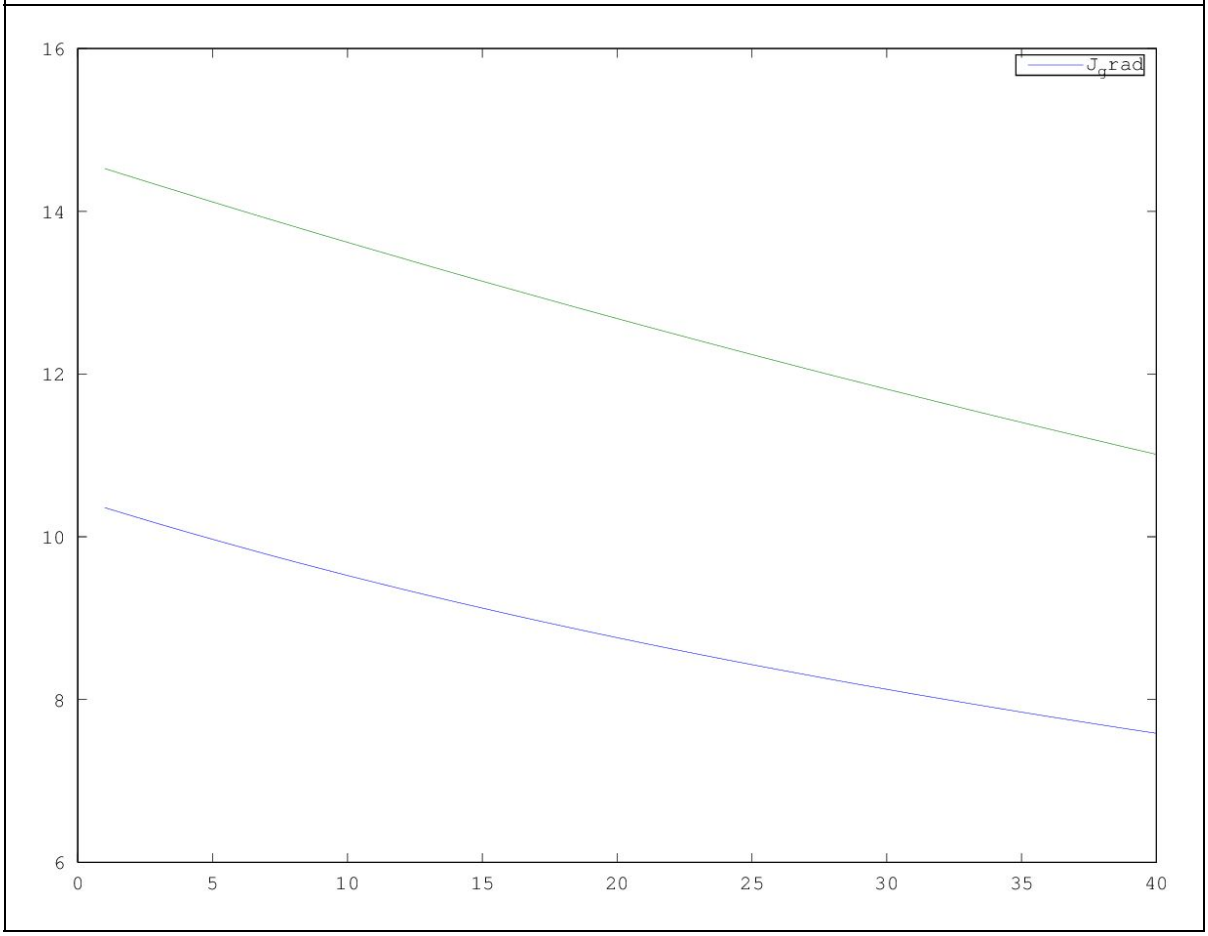
Tamaño dataset	Muestras	Características	Tiempo (segundos)	Verificación del gradiente
1x2	1	2	4.02410888671875	3.46597264757990e-10

Después de esta verificación se pasó a entrenar la red con 100 muestras.

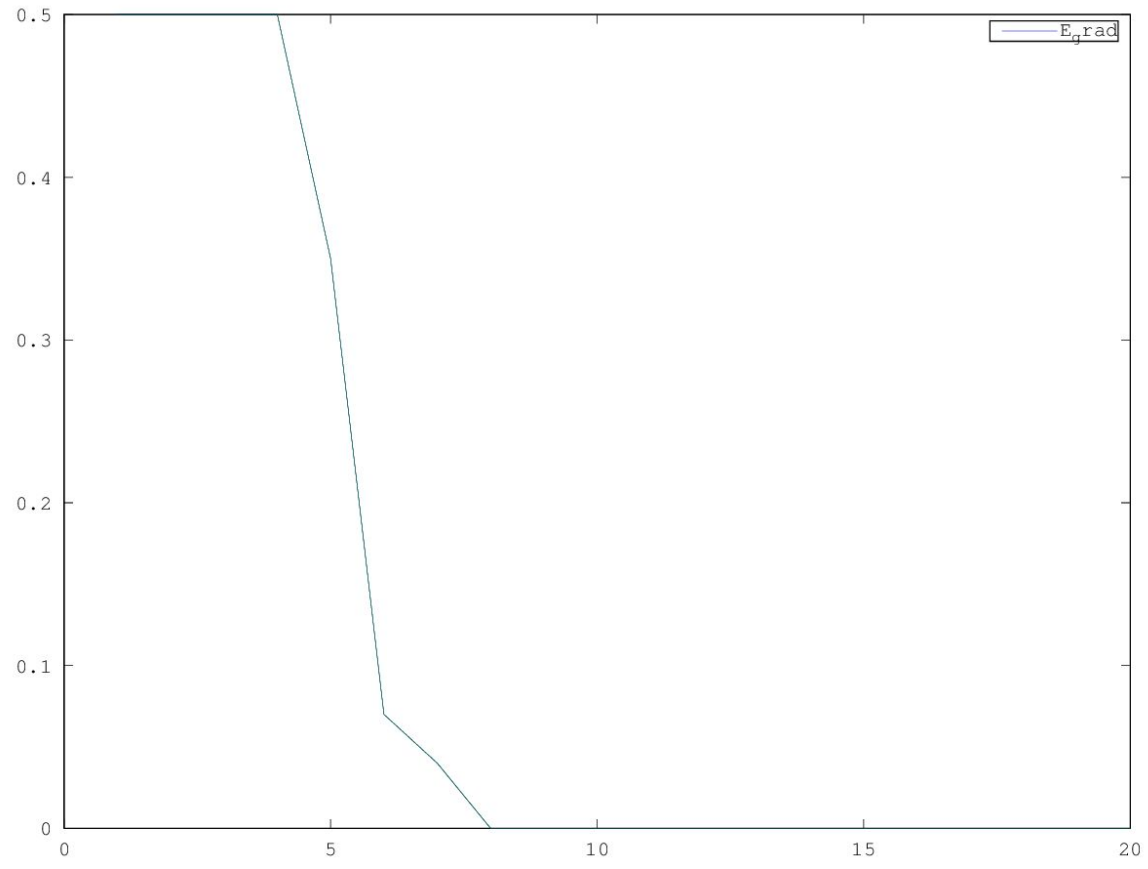
Tamaño dataset	Muestras	Características	Datos para entrenamiento (100%)
100 X 2	100	2	100

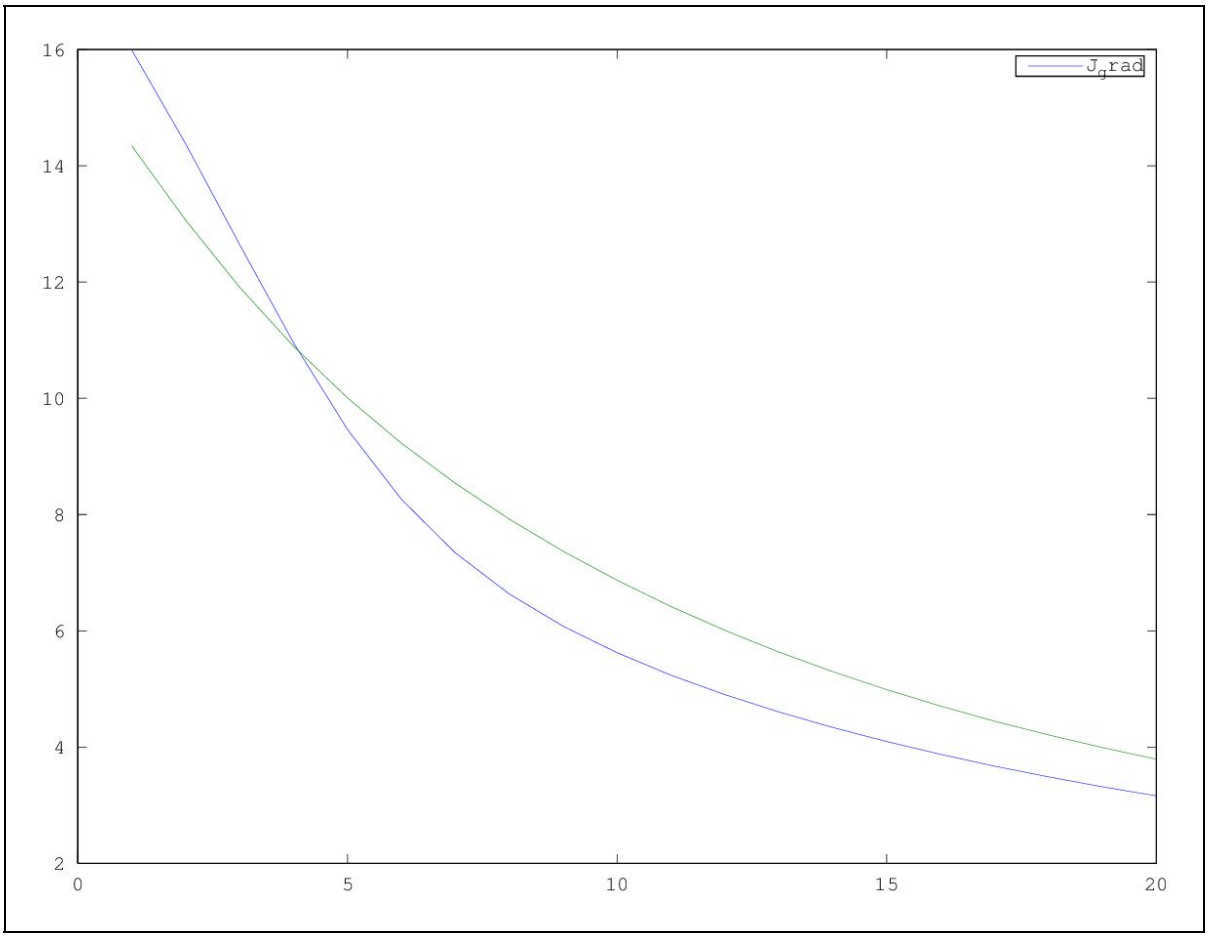
Neuronas capa entrada	Neuronas capa oculta	Neuronas capa salida
2	10	2
Tiempo (segundos)	Alpha	N# Iteraciones
4.13	0.001	40
Error= 0.08	Aciertos=92	Fallos=8


0.5
0.4
0.3
0.2
0.1
0
0
5
10
15
20
25
30
35
40
E_grad

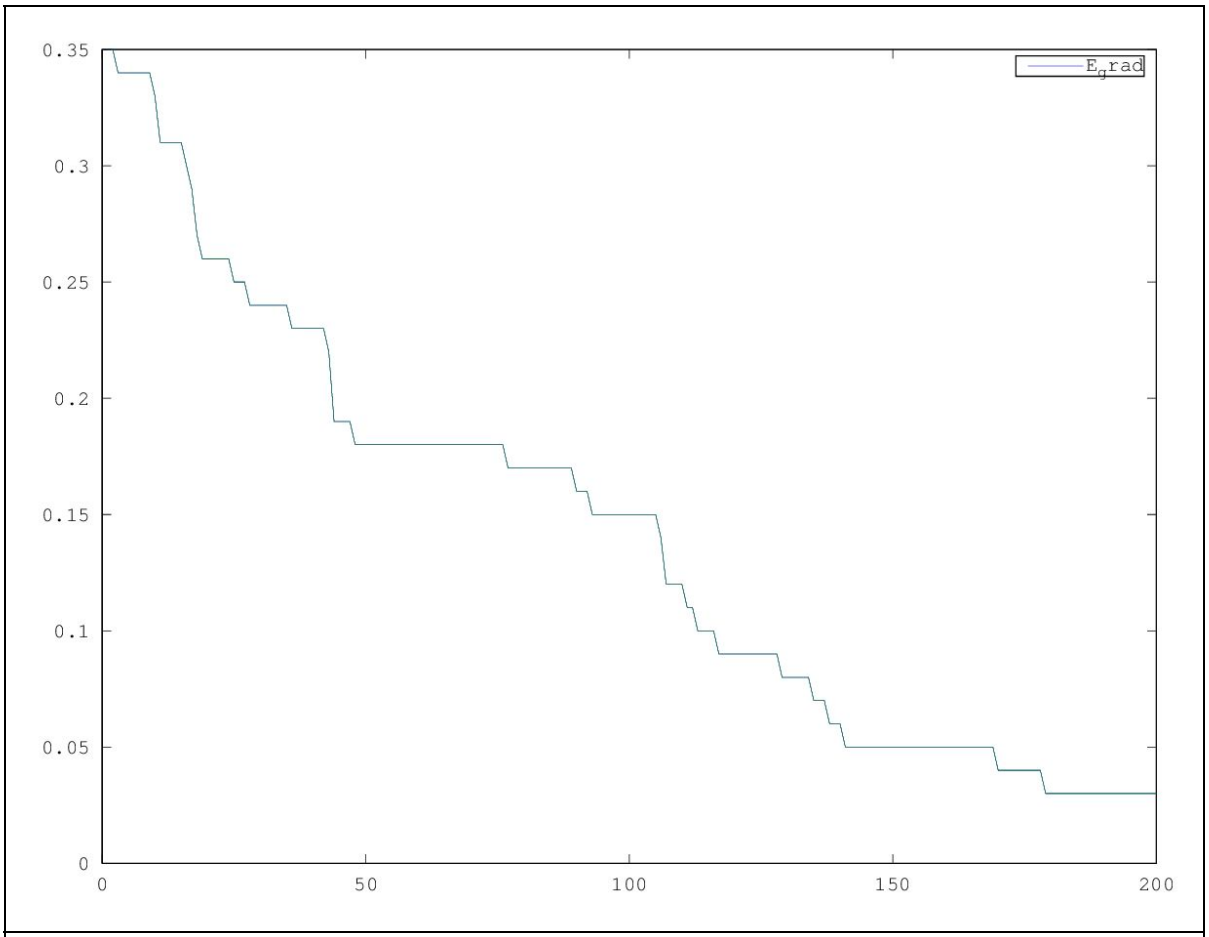

16
14
12
10
8
6
0
5
10
15
20
25
30
35
40
J_grad

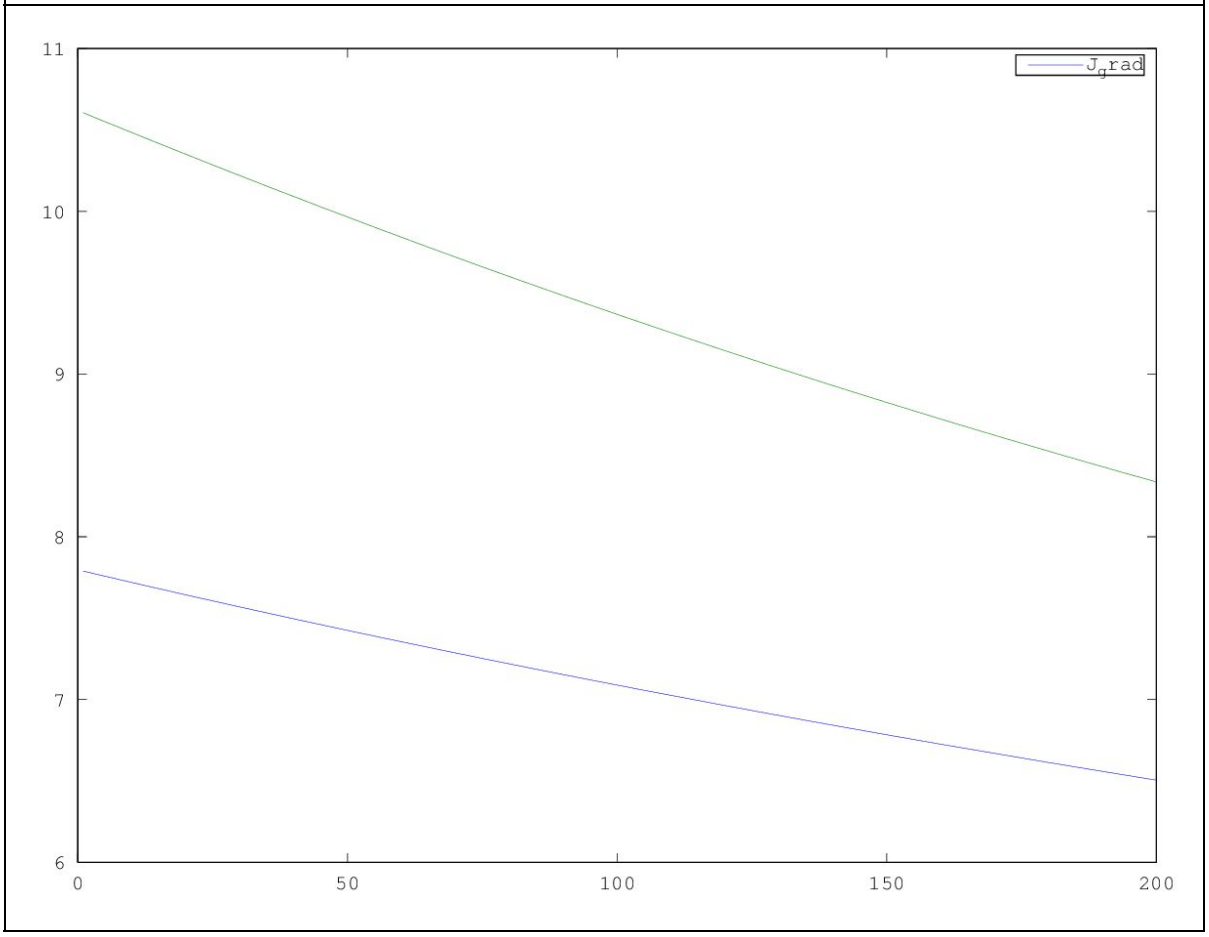
Neuronas capa entrada	Neuronas capa oculta	Neuronas capa salida
2	10	2
Tiempo (segundos)	Alpha	N# Iteraciones
2.44	0.01	20
Error= 0.0	Aciertos=100	Fallos=0



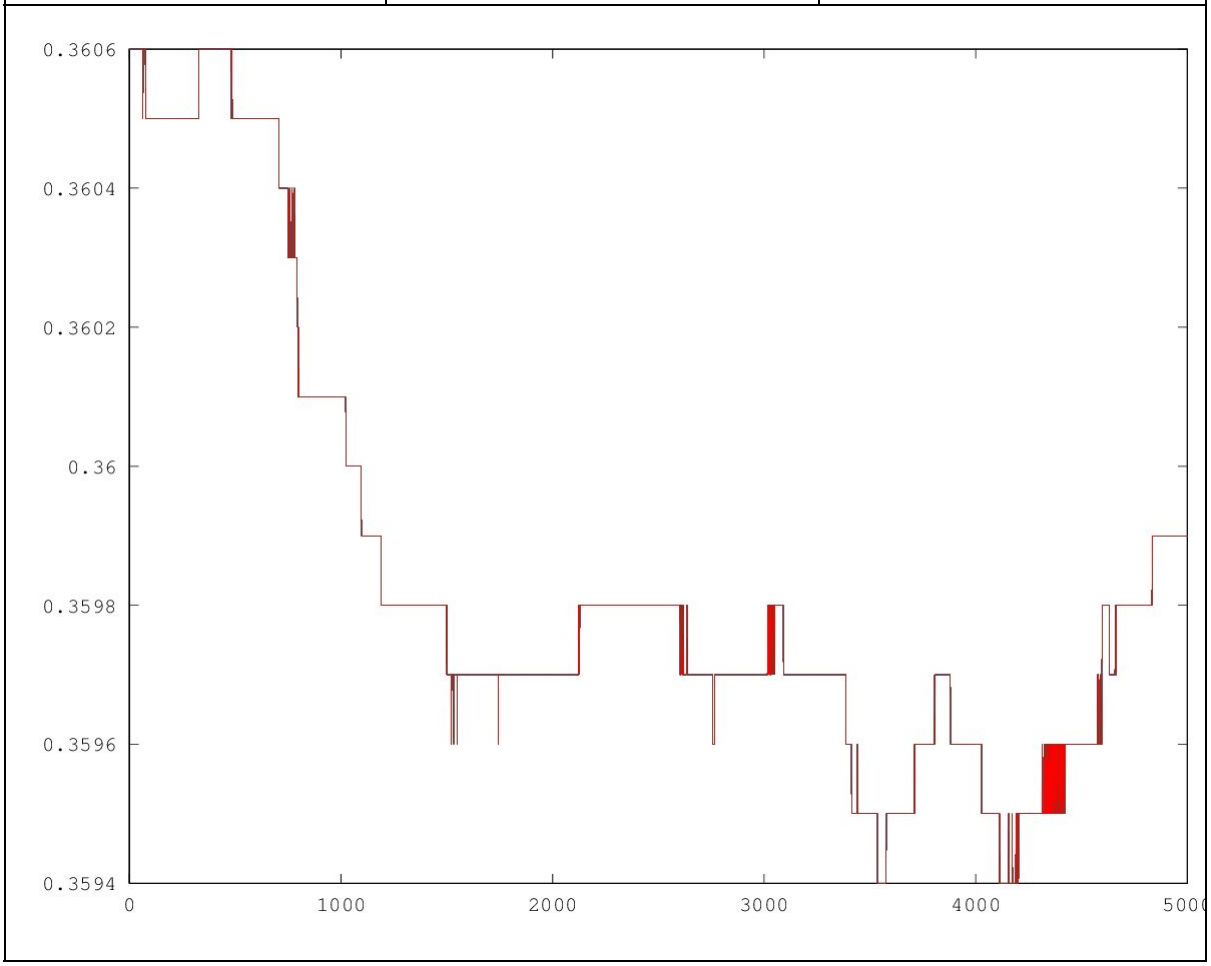


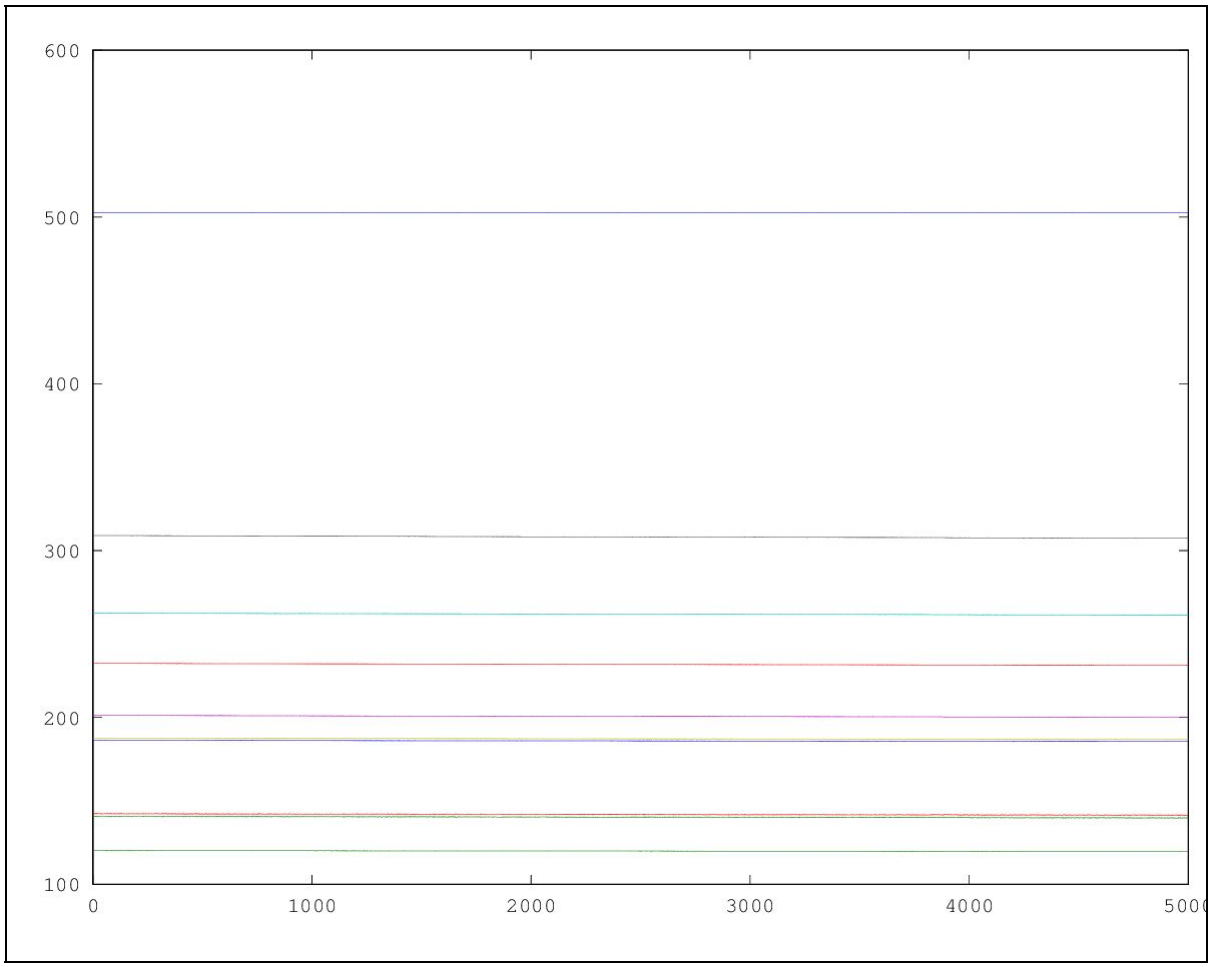
Neuronas capa entrada	Neuronas capa oculta	Neuronas capa salida
2	10	2
Tiempo (segundos)	Alpha	N# Iteraciones
7.49	0.0001	200
Error= 0.03	Aciertos=97	Fallos=3


0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
50
100
150
200
$E_g rad$

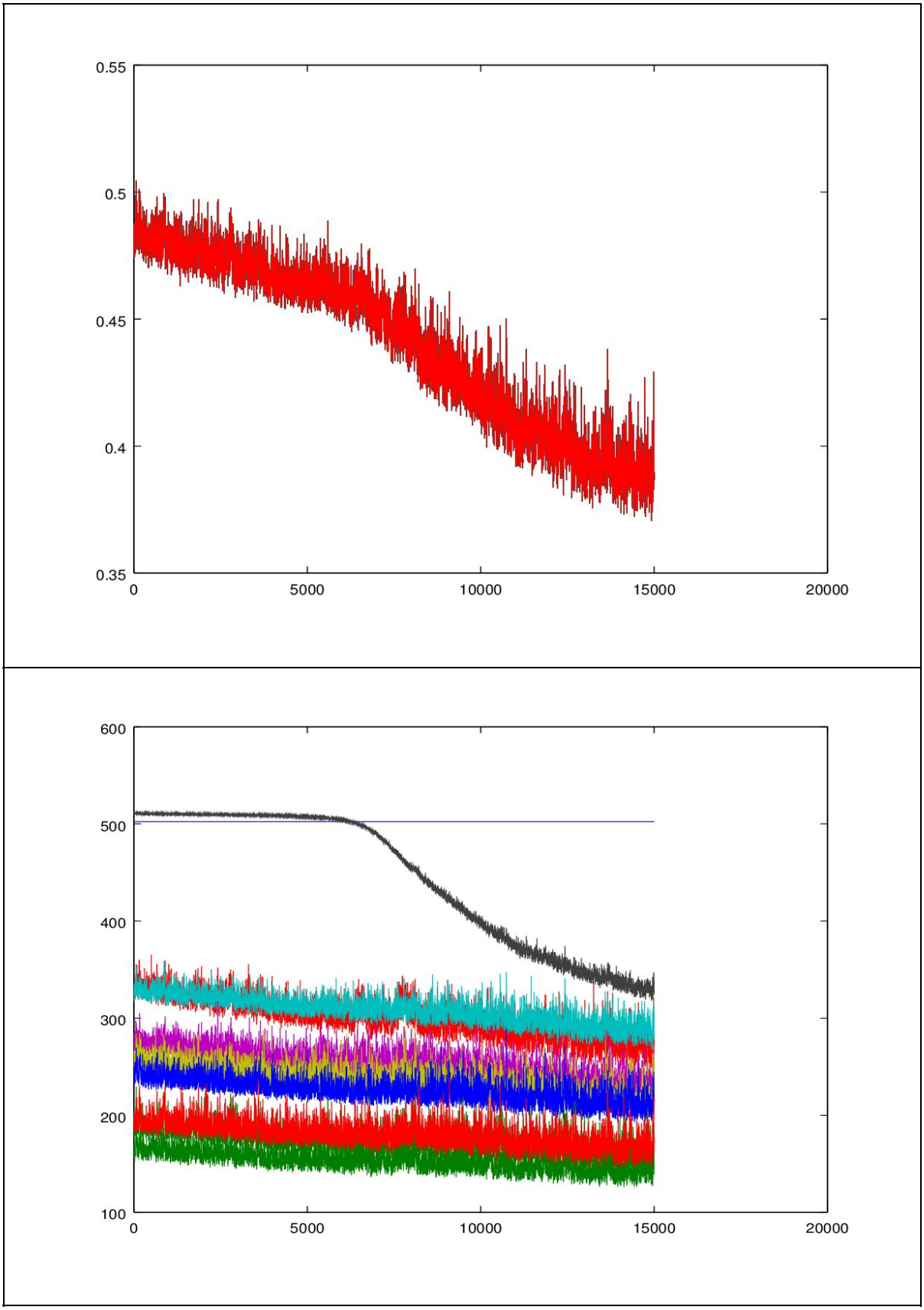

11
10
9
8
7
6
0
50
100
150
200
$J_g rad$

Neuronas capa entrada	Neuronas capa oculta	Neuronas capa salida
3074 x 1000	300	10
Tiempo (segundos)	Alpha	N# Iteraciones (Partiendo w_38E)
17400.3919982910	0.00001	5000
Error=0.36	Aciertos=6394	Fallos=3606





Neuronas capa entrada	Neuronas capa oculta	Neuronas capa salida
3074 x 1000	300	10
Tiempo (segundos)	Alpha	N# Iteraciones (Partiendo w_48E)
15299.8868942261	0.0001	15000
Error=0.388	Aciertos=6118	Fallos= 3882


0.55
0.5
0.45
0.4
0.35
0
5000
10000
15000
20000
600
500
400
300
200
100
0
5000
10000
15000
20000

Conclusiones :

1. En el proceso de aprendizaje con el dataset de 100 muestras, se pudo observar que cuando el alpha es más pequeño, son necesarias más iteraciones para que la red converja.
2. En el proceso de aprendizaje con el dataset de 10000 muestras, se pudo observar que cuando el alpha es grande, en la gráfica del error se ven variaciones muy bruscas.
3. Uno de los problemas más tediosos dentro a la elaboración del modelo, es la depuración de la red neuronal.