FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

86.54/66.63 - Redes Neuronales

Guía de ejercicios N°2

Rossi, Francisco 99540 frrossi@fi.uba.ar

26 de Enero de 2021

Índice

1. Ejercicio 1: Implementación de un `Perceptrón Simple` AND, OR de 2 y 4 Entradas **2**
1.1. Enunciado: . . . 2
1.2. Implementación . . . 2
1.2.a. 2 entradas . . . 2
1.2.b. 4 entradas . . . 3

2. Ejercicio 2: Implementación de un `Perceptrón Multicapa` XOR de 2 y 4 Entradas (Backpropagation) **4**
2.1. Enunciado: . . . 4
2.2. Implementación . . . 4
2.2.a. 2 entradas . . . 4
2.2.b. 4 entradas . . . 5

3. Ejercicio 3: Backpropagation y Perceptrón multicapa **6**
3.1. Enunciado: . . . 6
3.2. Implementación . . . 6

4. Ejercicio 4: Perceptrón Multicapa XOR 2 Entradas - Algoritmo Genético **8**
4.1. Enunciado: . . . 8

5. Ejercicio 5: Perceptrón Multicapa XOR 2 Entradas - Algoritmo Simulated Annealing **9**
5.1. Enunciado: . . . 9

6. Ejercicio Adicional: Capacidad del Perceptrón Simple **11**
6.1. Enunciado: . . . 11

7. Ejercicio Adicional: Boltzmann **12**

1. Ejercicio 1: Implementación de un `Perceptrón Simple` AND, OR de 2 y 4 Entradas

1.1. Enunciado:

Implemente un perceptrón simple que aprenda la función lógica AND de 2 y de 4 entradas. Lo mismo para la función lógica OR.

1.2. Implementación

1.2.a. 2 entradas

En el caso del perceptrón simple para una AND u OR de 2 entradas se tendran 3 pesos sinápticos, uno asociado a la primer columna de la tabla de verdad, otra a la segunda y por último el BIAS. La salida lineal de la neurona sera $h = w * x$ siendo w el vector de pesos sinápticos y x el vector de entradas con primer columna de unos para el bias.

El error seguirá siendo el error cuadrático medio por la cantidad de patrones a aprender sobre dos entre la salida actual y la deseada. En esta implmentación la función de activación será la función signo definida en la guía anterior.

Se genera un vector inicial de pesos sinápticos w y se procede a calcular el error para estos pesos. Luego se itera hasta que se tenga error nulo, es decir la salida de la red es exactamente la salida deseada.

Se realiza este proceso para la AND y la OR de dos entradas y se obtuvo el siguiente resultado final en las **Figs. 1 y 2**, donde se comprueba que ambas son problemas linealmente separables, las ecuaciónes de la rectas son:

$$y = -\frac{w_1}{w_3} - \frac{w_3}{w_2}x$$

Siendo y una entrada e x la otra.

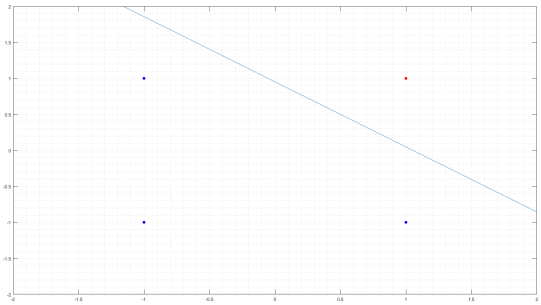

Figura 1: Resultado del entrenamiento del perceptrón simple con AND

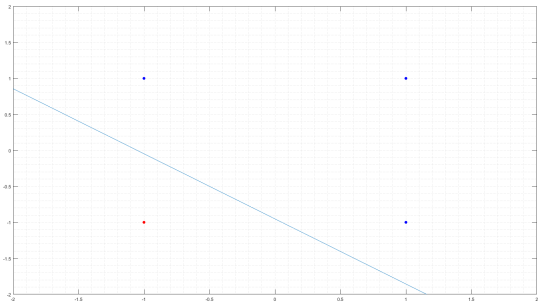

Figura 2: Resultado del entrenamiento del perceptrón simple con OR

También se muestra el error en función de las iteraciones en las **Figs. 3 y 4** a continuación.

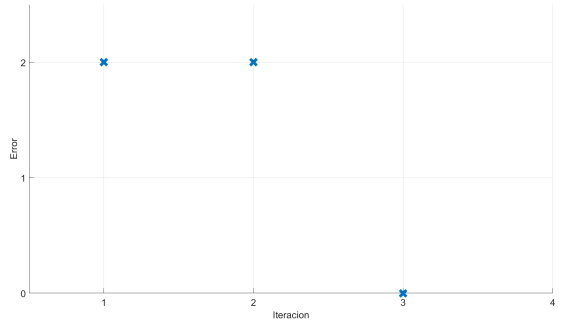

Figura 3: Error en función de la cantidad de iteraciones AND

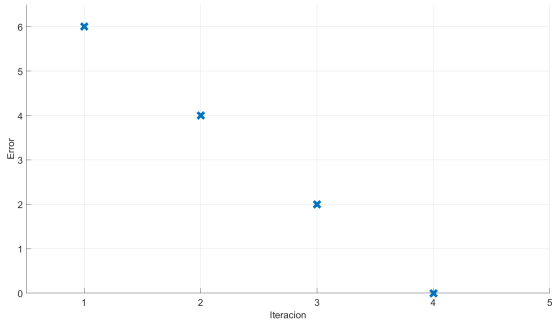

Figura 4: Error en función de la cantidad de iteraciones OR

1.2.b. 4 entradas

Para el caso de 4 entradas se realiza el mismo proceso, pero al no poder representarse en un gráfico de 2-D se muestran los gráficos del error en función de las iteraciones en las **Figs. 5 y 6** .

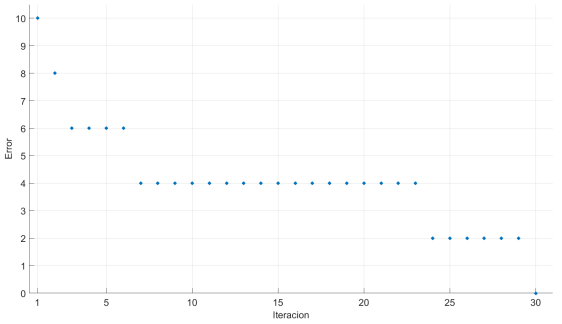

Figura 5: Error en función de la cantidad de iteraciones AND de 4 entradas

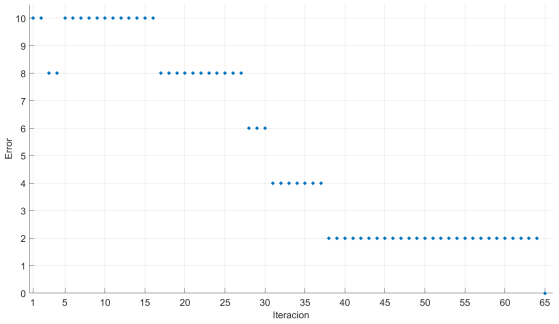

Figura 6: Error en función de la cantidad de iteraciones OR de 4 entradas

2. Ejercicio 2: Implementación de un `Perceptrón Multicapa` XOR de 2 y 4 Entradas (Backpropagation)

2.1. Enunciado:

Implemente un perceptrón multicapa que aprenda la función lógica XOR de 2 y de 4 entradas (utilizando el algoritmo `backpropagation`)

2.2. Implementación

En este caso el problema XOR no es linealmente separable de manera que utilizando un perceptrón simple no se podría resolver. Por eso se utiliza el perceptrón multicapa, en este caso de 2 capas ocultas y 4 neuronas por capa. Se inicializan los pesos sinápticos iniciales con variables normales $\mathcal{N} \sim (0, 0{,}5)$. Se utiliza en este caso la función de activación tanh pues para utilizar el algoritmo `backpropagation` debemos utilizar una función derivable.

2.2.a. 2 entradas

Se computa el error en cada iteración hasta llegar a un error máximo aceptable y se obtiene el gráfico de la **Fig. 7**.

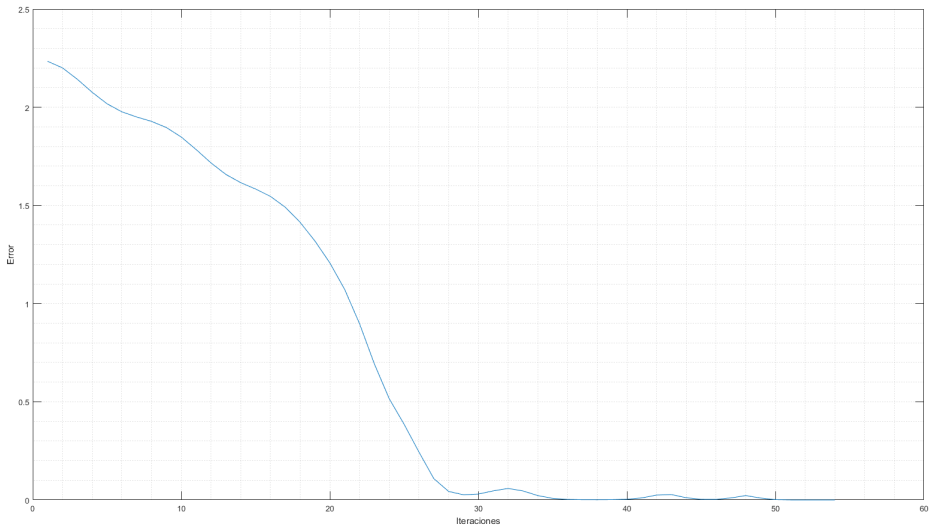

Figura 7: Error en función de las iteraciones 2 entradas

Se obtuvieron los siguientes valores de salida.

Tabla 1: Salida de la red para XOR de 2 entradas

	y_d	y
1	-1	-1
2	1	1
3	1	0.9994
4	-1	-0.999

Con un error de $\boxed{E = 2{,}048 \cdot 10^{-7}}$

2.2.b. 4 entradas

Se computa el error en cada iteración hasta llegar a un error maximo aceptable y se obtiene el gráfico de la **Fig. 7**.

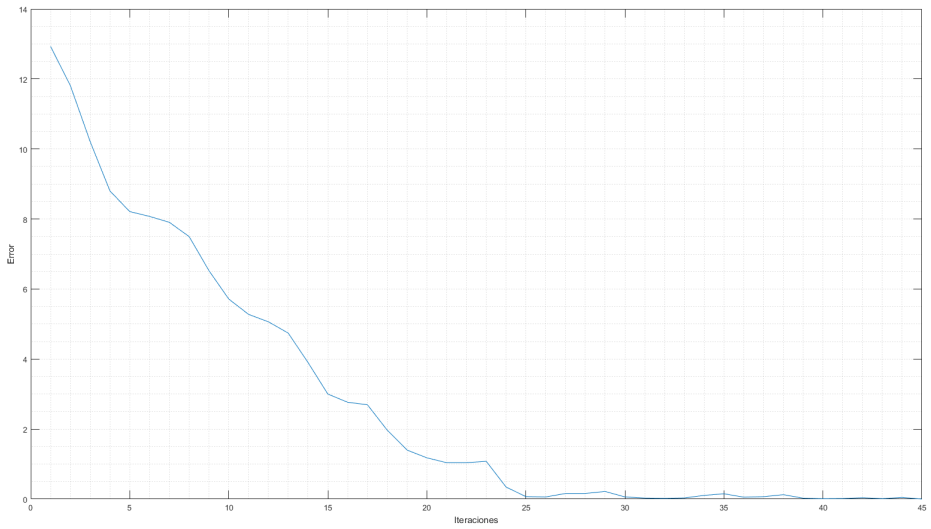

Figura 8: Error en función de las iteraciones 4 entradas

Se obtuvieron los siguientes valores de salida.

Tabla 2: Salida de la red para XOR de 4 entradas

	y_d	y
1	-1	-1
2	1	1
3	1	1
4	-1	-1
5	1	0.9995
6	-1	-1
7	-1	-1
8	1	1
9	1	0.9988
10	-1	-1
11	-1	-1
12	1	1
13	-1	-1
14	1	1
15	1	1
16	-1	-1

Con un error de $\boxed{E = 8{,}684 \cdot 10^{-7}}$

3. Ejercicio 3: Backpropagation y Perceptrón multicapa

3.1. Enunciado:

Implemente una red con aprendizaje Backpropagation que aprenda la siguiente función

$$f(x, y, z) = \sin(x) + \cos(y) + z$$

donde: $x, y \in (0, 2\pi]$ y $z \in [-1, 1]$

3.2. Implementación

Esta red se arma con una capa de entrada y otra de salida con 32 neuronas cada una. para poder utilizar tanh como función de activación la salida con la cual se comparara será $f(x, y, z)/3$ x, y ,z se generan aleatoriamente con $N = 1000$ uniformes para el entrenamiento (train) y $M = 400$ para el testeo (test).

Se entrena la red mediante backpropagation hasta alcanzar el máximo error aceptado o alcanzar las 500 épocas. El error se normaliza con respecto a la cantidad de patrones enseñados/testeados.

En la **Fig. 9** vemos el error E_{train} en función de las iteraciones, es decir cada vez que se actualizan los pesos. En la **Fig. 10** se muestra el error E_{test} en función de cada época, cada vez que se recorren todos los patrones del vector de entrenamiento y se testea contra el vector de testing.

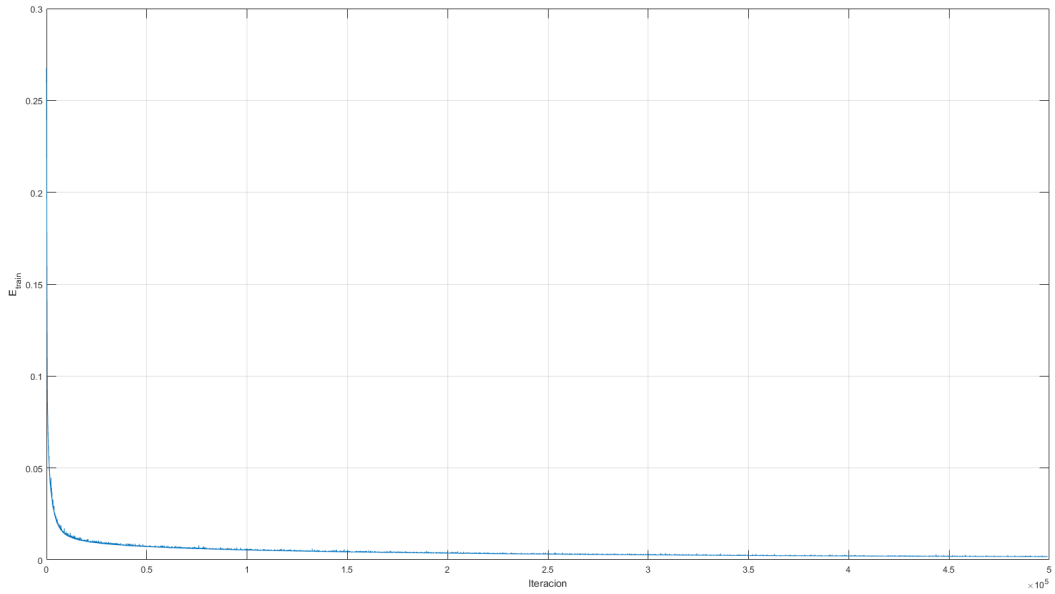

Figura 9: Error en función de cada iteración (actualización)

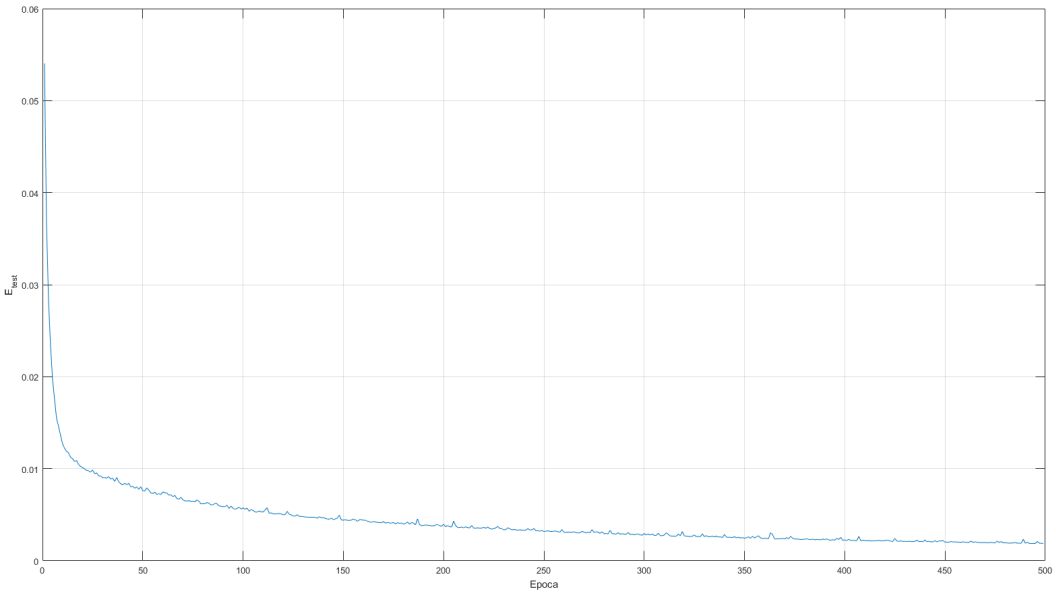

Figura 10: Error en función de cada época

Luego para generar un gráfico ameno para comparara la salida con la salida deseada se realizo cuadrados mínimos con y_d Vs. $y_o ut$ salida deseada Vs. salida. De manera de que cuanto más cercano a la recta $y_d = y_o ut$ menor error.

La recta obtenida tiene la forma:

$$y_{out} = 0{,}96829 \cdot y_d + 0{,}0028587$$

Como puede observarse en la **Fig. 11**

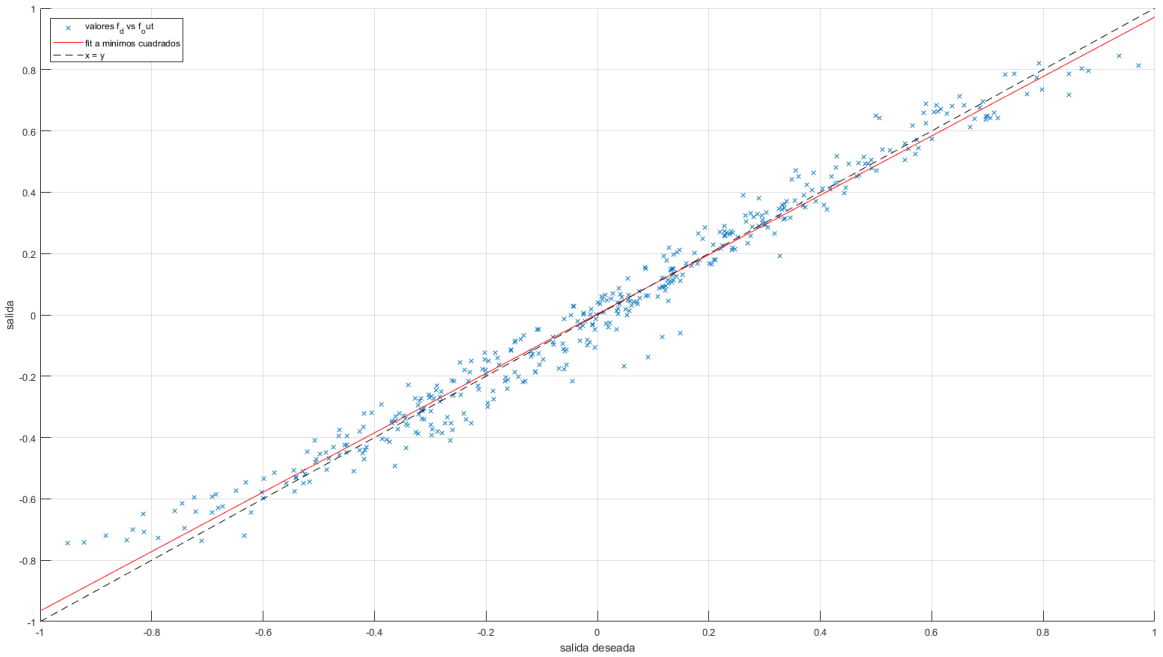

Figura 11: Salida deseada en función de la salida

4. Ejercicio 4: Perceptrón Multicapa XOR 2 Entradas - Algoritmo Genético

4.1. Enunciado:

Encontrar un perceptrón multicapa que resuelva una XOR de 2 entradas con un algoritmo genético.

En este ejercicio se implementará nuevamente una XOR de 2 entradas, el aprendizaje estará realizado con el algoritmo genético. El cuál consiste en entrenar multiples redes de perceptrones multicapa en este caso con 1 capa oculta de 4 neuronas cada capa y 11 redes. Luego de inicializarlas con la función `crear_individuo` que no hace nada más que generar una matriz de pesos sinápticos donde cada columna representan los pesos sinápticos para cada red y se evalua el fitness que va entre [0,1] de cada individuo (red):

$$F = E/4 \tag{1}$$

Donde E es el error que venimos utilizando en ejercicios anteriores. Como antes se seteaba un error máximo en este caso buscaremos un valor de fitness mínimo para cortar el algoritmo $fitness\_ths = 0{,}99$.

A partir de la generación inicial de la población se calcula el fitness y se procede a buscar siempre el elite, es decir el individuo con mayor fitness. Luego mientras no exista un individuo con $fitness \geq fitness\_ths$ se procede a realizar el algoritmo.

Se busca formar una nueva generación de la población. El Elite si o si deja herencia directamente y por ende lo copiamos a la nueva generación. Luego se eligiran pares de la población para reproducirse, para esto se determina un intervalo entre [0,1] a cada individuo que es de longitud igual a su probabilidad de ser elegidos: $p_{rep} = \frac{F}{\sum_i F_i}$ de esta manera se realiza una multinomial para elegir estos individuos con el vector de probabilidad generado a partir de cada una de estas probabilidades $\frac{F_i}{\sum_i F_i}$, si sale dos veces el mismo se vuelve a elegir. Luego, si uno de los dos es elite se copian ambos directamente a la siguiente generación, si no, se da un crossover con probabilidad $p_{cross} = 0{,}05$ en cual se intercambian una serie de pesos sinápticos entre ambas redes.

El siguiente paso es la mutación, una vez obtenidos los 11 individuos de la (casi) nueva generación se procede a mutar los mismos sumando a los pesos una VA normal, excepto el elite que ha sido copiado, ese debe quedar idéntico. Luego se vuelve a computar el fitness y se repite este proceso hasta alcanzar al menos 1 individuo que cumpla el fitness propuesto.

En el gráfico de la **Fig. 12** se muestra el fitness del individuo elite en cada generación.

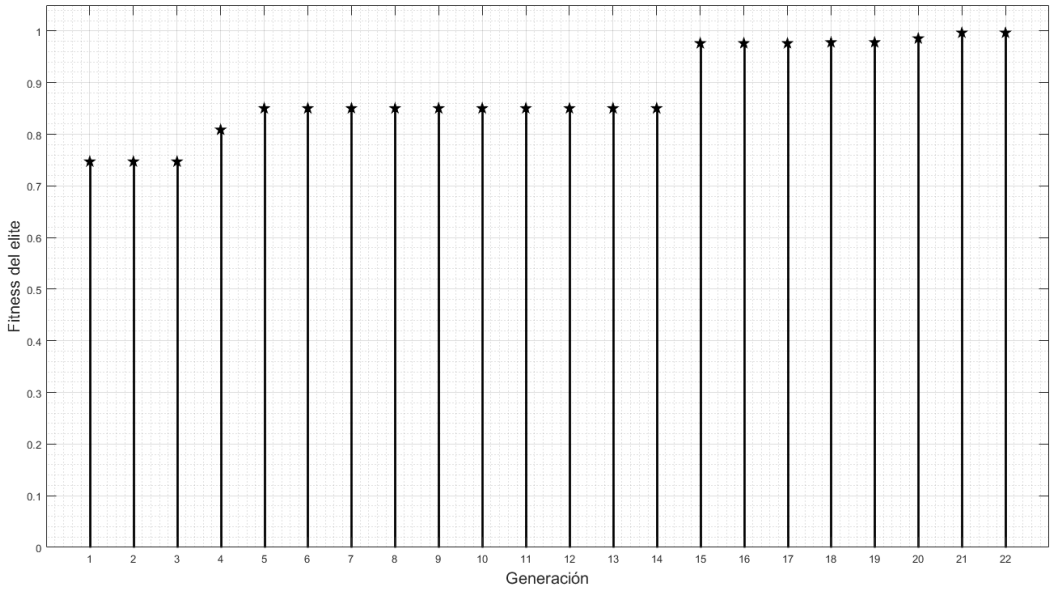

Figura 12: Fitness del individuo elite en función de la generación

5. Ejercicio 5: Perceptrón Multicapa XOR 2 Entradas - Algoritmo Simulated Annealing

5.1. Enunciado:

Encontrar un perceptrón multicapa que resuelva una XOR de 2 entradas mediante simulated annealing.

En este ejercicio se implementará nuevamente una XOR de 2 entradas, en este caso el aprendizaje se realizará mediante simulated annealing, de la siguiente manera.

La inicialización de los pessos es identica a ejercicios anteirores, con normales $\mathcal{N}(0,\sigma)$ proponiendo un error máximo aceptable de $E_target = 1 \cdot 10^{-4}$. Luego se recorre una vez la red para obtener un primer valor de error. Luego mientras el error no sea menor igual a E_target se realizará el siguiente algoritmo.

A lo largo del mismo se ira decrementando la temperatura de forma

$$T = \alpha \cdot T \quad , \alpha = 0{,}99$$

1. $W = W + \mathcal{N}(0, var)$ se modifican los pesos (a una variable auxiliar)
2. Se computa el nuevo error E
3. Si $E_{actual} < E_{anterior}$ se modifican los pesos y se acepta el cambio.
4. Si $E_{actual} \geq E_{anterior}$ se acepta el cambio con probabilidad $p = e^{-\frac{E_{anterior} - E_{actual}}{k * T}}$, si se rechaza no se modifican los pesos y se vuelve al estado anterior.
5. Se decrementa la temperatura.

Si llega a la temperatura mínima antes de llegar a un error menor el algoritmo no convergió.

Se realizo esto con varianza 1 y 0.5 y se obtuvieron los gráficos de las **Figs. 13 y 13** respectivamente.

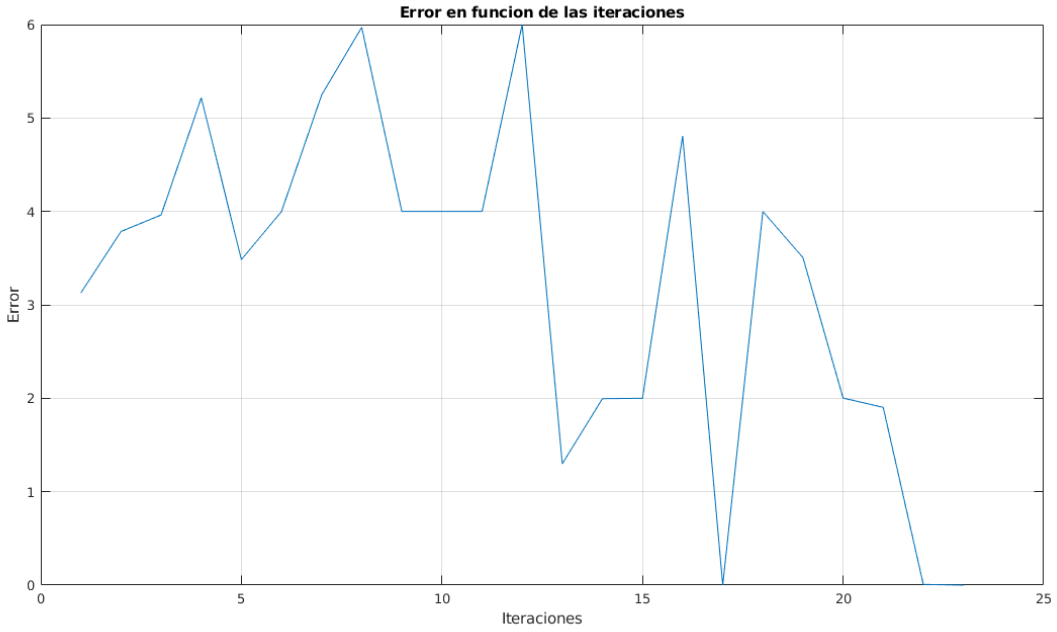

Figura 13: Error en función de cada iteración $\sigma = 1$

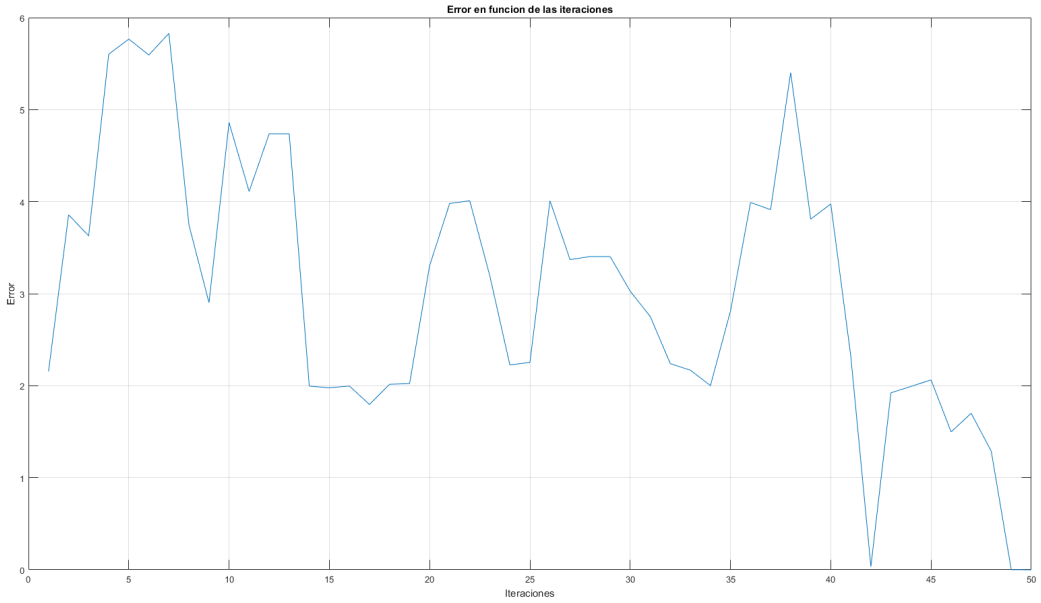

Figura 14: Error en función de cada iteración $\sigma = 0{,}5$

Donde podemos apreciar que cuanto menor es la varianza más iteraciones son necesarios para que el método converja, pues en cada iteración los cambios son menores.

6. Ejercicio Adicional: Capacidad del Perceptrón Simple

6.1. Enunciado:

Hallar la capacidad para una `Perceptron Simple`

En este caso se le enseñan patrones aleatorios y se evalua si se aprendio o no el patron, incrementando la cantidad de patrones enseñados. Luego se gráfica la cantidad de patrones aprendidos sobre los enseñados en función de la cantidad de patrones enseñados como se oberva en la **Fig. 15** hasta $p = 20$ patrones la red aprende todos, luego va decayendo hasta no aprender ninguno pasando los $p = 50$ patrones. Por ejemplo con $p = 26$ patrones, la red aprende $0{,}9 \cdot 26 = 23{,}4$ patrones. Se utilizaron patrones de largo $N = 20$.

Al llegar a el doble de patrones que la longitud del vector de entradas la capacidad debería ser cercana a 0.5, ya que si tuviéramos un vector de entrada infinito el gráfico sería una función escalón $u(t - 2N)$, es decir puede aprender 2N patrones, siendo N el largo de la entrada. Sin embargo, en la **Fig. 15** vemos que la capacidad llega a la mitad en aproximadamente $p = 35$, a medida que se aumente la cantidad de entradas debería ser cada vez más similar al escalón descripto anteriormente.

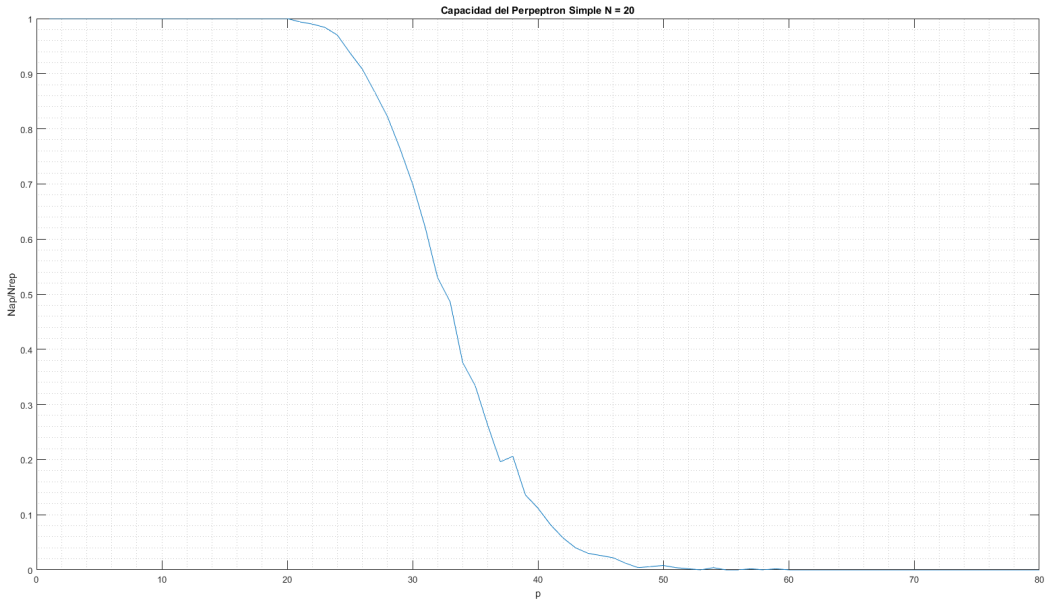

Figura 15: Capacidad de un perceptrón simple

7. Ejercicio Adicional: Boltzmann

En este ejercicio se busca implementar una máquina de Boltzmann restringida. En la misma tenemos ciertas neuronas en la capa visible (784) dado que las imágenes de entrada son de 28x28. Y 10 neuronas en la capa oculta. Se inicializa la matriz de pesos con normales estándar, al igual que los bias de ambas capas. Se la entrena con 5000 mil iteraciones sobre el vector de entrenamiento, con constante de aprendizaje $\epsilon = 0{,}1$ y divergencia contrastiva de orden 1. Se actualizan los pesos en cada iteración hasta entrenar a la red.

En resumen al ingresar una imagen a la red `m_recon` esta pasa a la capa oculta con la matriz de pesos W y sumado al bias de la capa oculta obtenemos la salida lineal del nodo de la capa oculta y a partir de una bernoulli de probabilidad $p = \frac{1}{1+e^{-x}}$ se define el valor de la capa oculta. Luego se reconstruye esta entrada con la matriz de pesos transpuesta sumada a los bias de la capa visible, obteniendo así la reconstrucción `m_recon`. Se computa el error en cada iamgen y luego de pasar por todas las imágenes se saca el valor medio de esta corrida para obtener el gráfico de la **Fig. 16** y se calculan las variaciones de los pesos y los bias, se actualizan los mismos y se vuelve a iterar hasta 5000 iteraciones.

El aprendizaje a medida que aumentan las iteraciones, se observa en la **Fig. 16** como disminuye el error medio. Esto verifica el aprendizaje de la red.

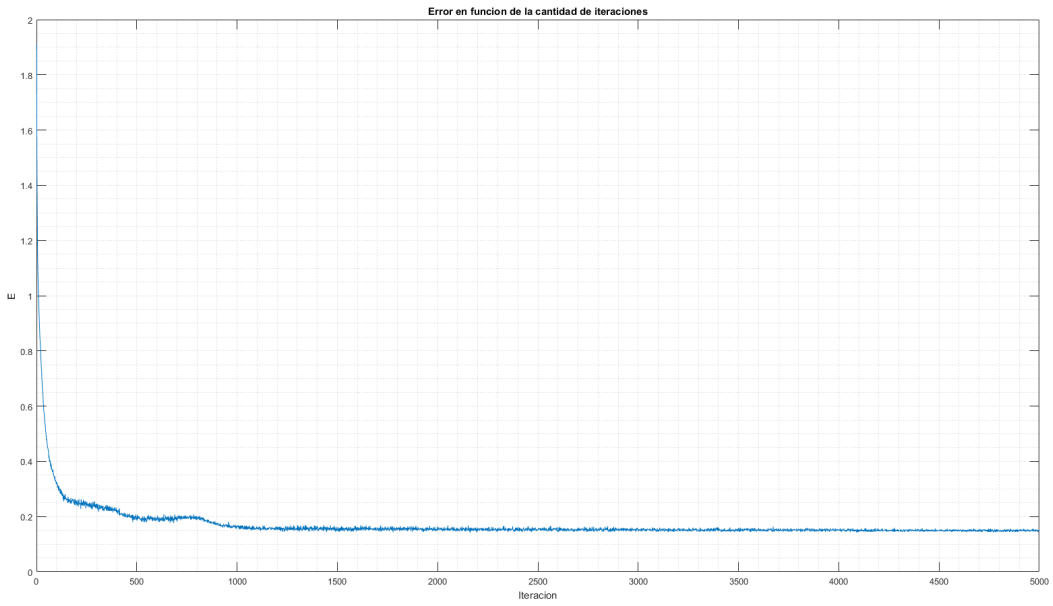

Figura 16: Error en función de la cantidad de iteraciones

A continuación de muestran algunos ejemplos de la red en algunos no la reconstruye a la perfección, pero el dígito es correcto como en las **Figs. 17, 18** o **Figs. 19, 20**, en otros confunde el dígito como en la **Figs. 23, 24** o **Figs. 21, 22**

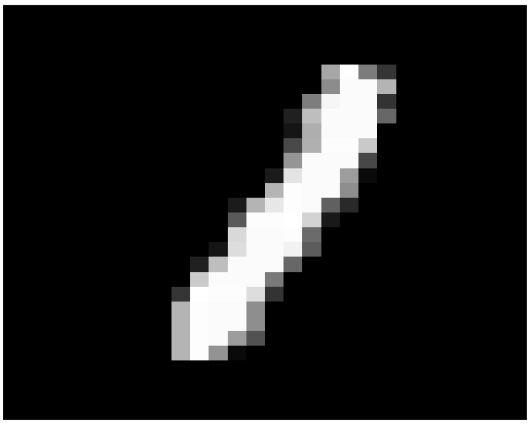

Figura 17: Imagen de entrada

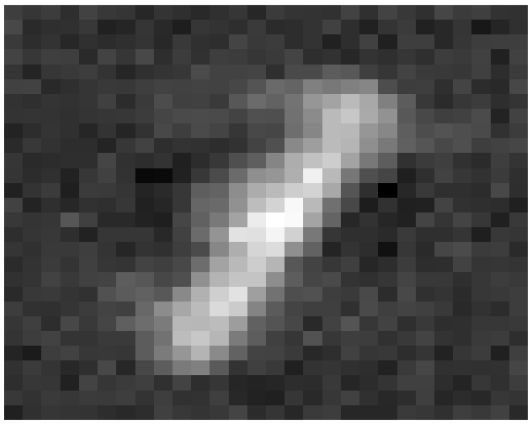

Figura 18: Imagen de salida

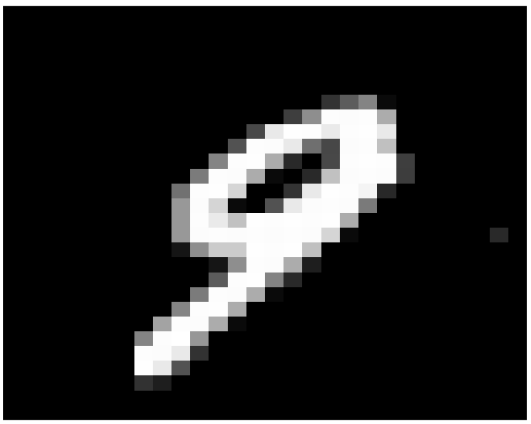

Figura 19: Imagen de entrada

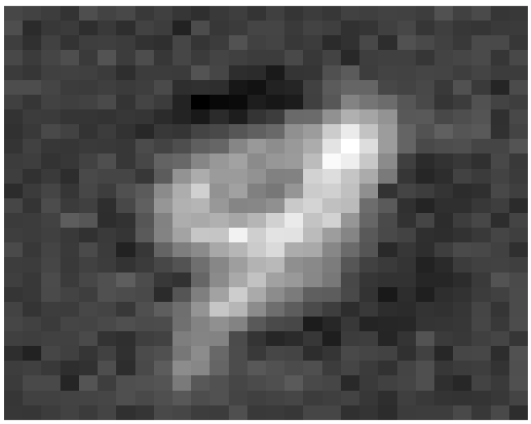

Figura 20: Imagen de salida

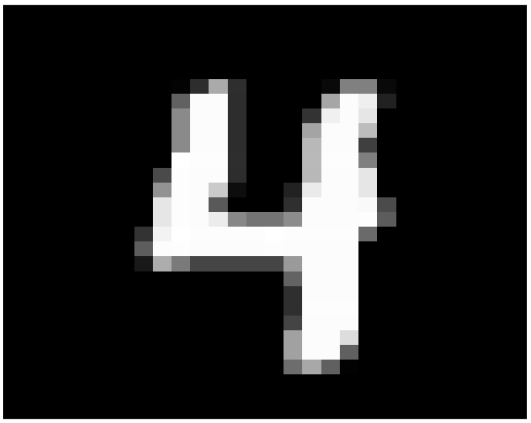

Figura 21: Imagen de entrada

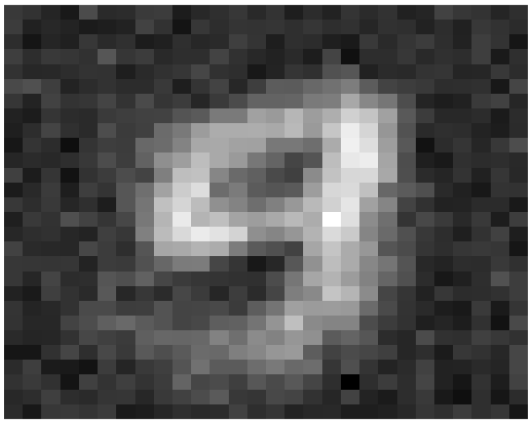

Figura 22: Imagen de salida

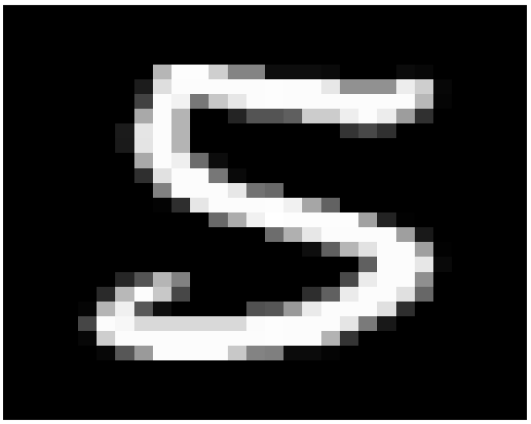

Figura 23: Imagen de entrada

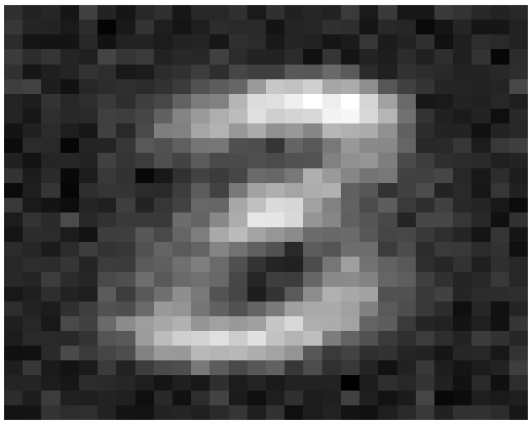

Figura 24: Imagen de salida