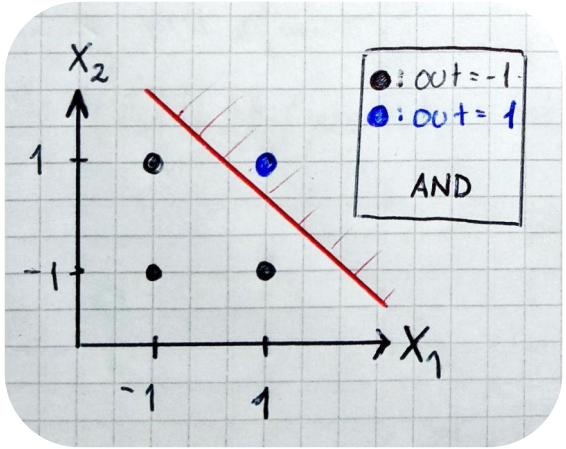
Sistemas de inteligencia artificial
TP3: Perceptron simple y multicapa

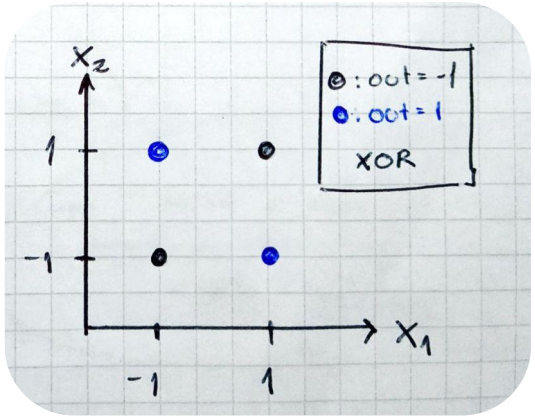
Grupo 19
Integrantes:
- ➔ Lucas Catolino
- ➔ Matias Ricarte

Sobre el perceptrón simple escalón

- El perceptrón simple escalón es útil a la hora de resolver problemas de separabilidad lineal.

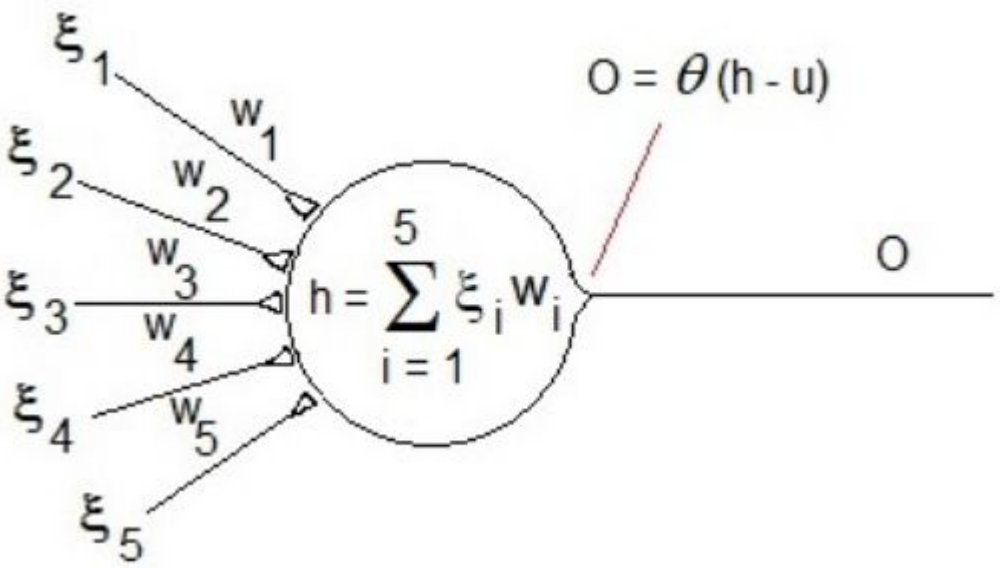
- XOR **no** es parte de ese conjunto de problemas.





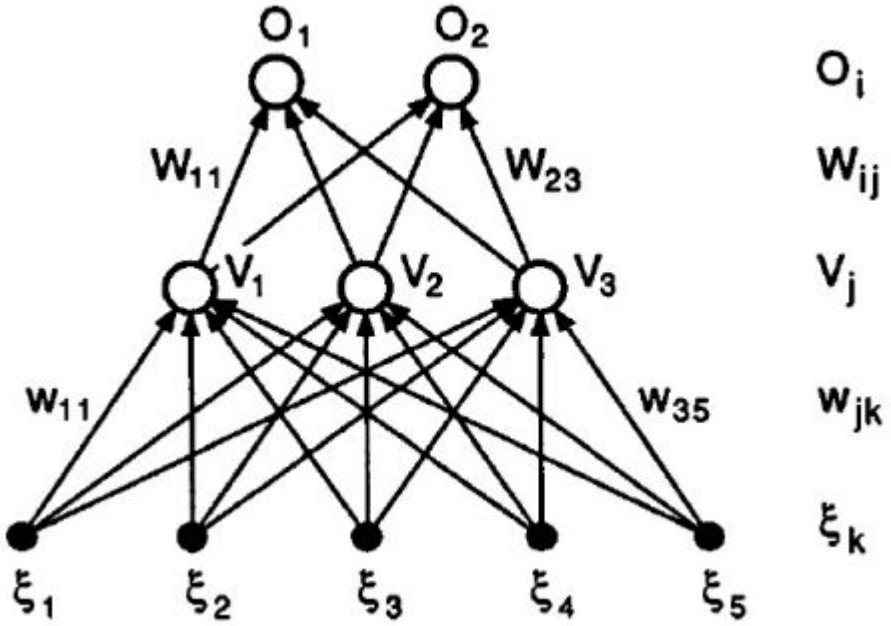
Perceptrón simple: modelo y algoritmo

```
i = 0
w = zeros(N+1, 1)
error = 1
error_min = p * 2
while error > 0 ∧ i < COTA
    Tomar un número i_x al azar entre 1 y p
    Calcular la exitación h = x[i_x].w
    Calcular la activación O = signo(h)
    Δw = η * (y[i_x] − O).x[i_x]
    w = w + Δw
    error = CalcularError(x, y, w, p)
    if error < error_min
        error_min = error
        w_min = w
    end
    i = i + 1
end
```

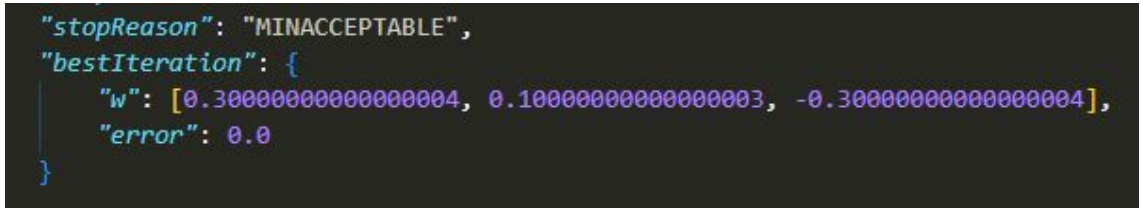


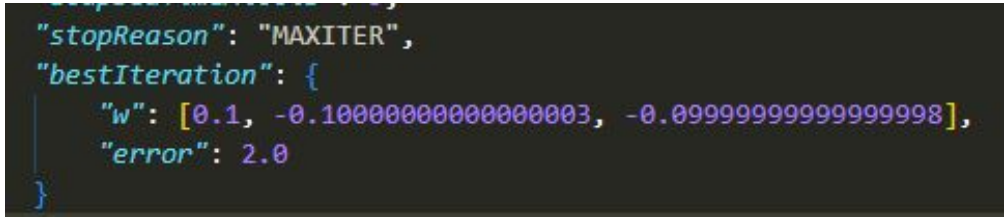
Perceptrón múltiple: modelo y algoritmo

1.Inicializar el conjunto de pesos en valores 'pequeños' al azar.
2.Tomar un ejemplo ξ^μ al azar del conjunto de entrenamiento y aplicarlo a la capa 0:
$V_k^0 = \xi_k^\mu$ para todo k.
3.Propagar la entrada hasta a capa de salida
$V_i^m = g(h_i^m) = g(\sum_j w_{ij}^m V_j^{m-1})$ para todo m desde 1 hasta M.
4.Calcular δ para la capa de salida
$\delta_i^M = g'(h_i^M)(\zeta_i^\mu - V_i^M)$
5.Retropropagar δ_i^M a la capa $m-1$
$\delta_j^{m-1} = g'(h_j^{m-1}) \sum_i w_{ij}^m \delta_i^m$ para todo m entre M y 2
6.Actualizar los pesos de las conexiones de acuerdo
$w.nuevo_{ij}^m = w.viejo_{ij}^m + \Delta w_{ij}^m$
donde $\Delta w_{ij}^m = \eta \delta_i^m V_j^{m-1}$
7.Calcular el *error*. Si $error > COTA$, ir a 2.



Resultados para el primer ejercicio

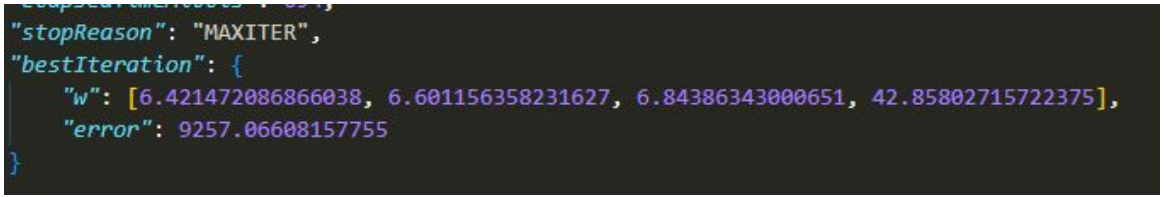


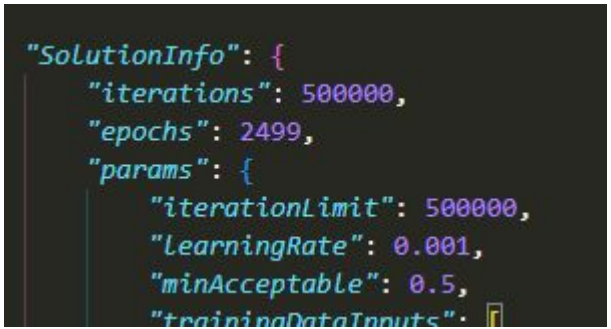


Sobre el perceptrón simple lineal y no lineal

Para el ejercicio 2 se nos dio un dataset de entradas y salidas para probar con estos 2 tipos de perceptrones simples.

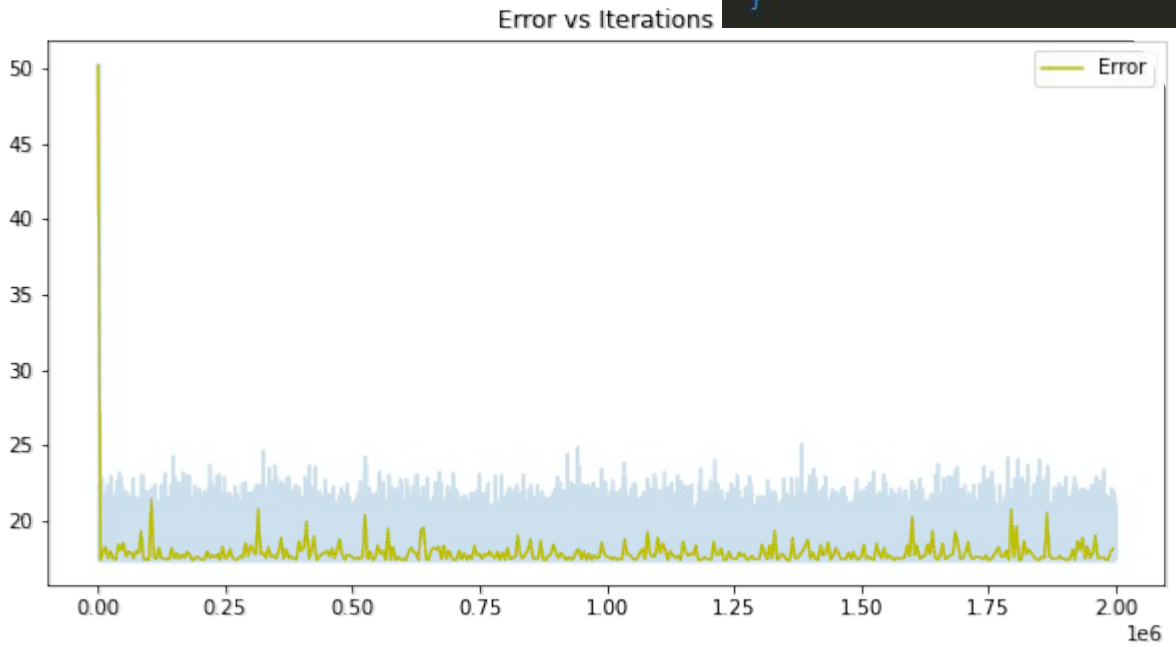
- En el lineal no fue posible bajar el error a un valor apropiado para decir que la red “aprendió” cómo resolver el problema. Esto nos lleva a pensar que la función a aprender **no** es una transformación lineal.





(cont.)


```
"elapsedTimeMillis": 34217,
"stopReason": "MAXITER",
"bestIteration": {
    "w": [0.23346248990706792, 0.2662405177410216, -0.21138101169215484],
    "error": 17.38400292356527
}
```


```
"SolutionInfo": {
    "iterations": 2000000,
    "epochs": 9999,
    "params": {
        "iterationLimit": 2000000,
        "learningRate": 0.01,
        "minAcceptable": 0.0,
```

```
"trainingDataInputSize": 200,
"perceptronMode": "nonlinear",
"kCuts": 4,
"beta": 0.8,
"sigmoidType": "tanh",
"printHistory": "both"
```

Aplicando validación cruzada en *k* partes

```
"iterationLimit": 2000000,
"learningRate": 0.01,
```

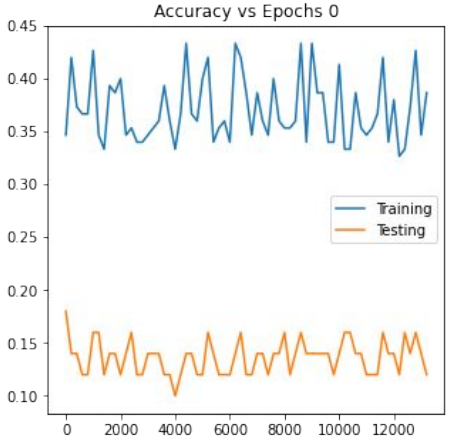
```
"perceptronMode": "nonlinear",
"sigmoidType": "tanh",
"beta": 0.8,
"printHistory": "both",
"kCuts": 4
```

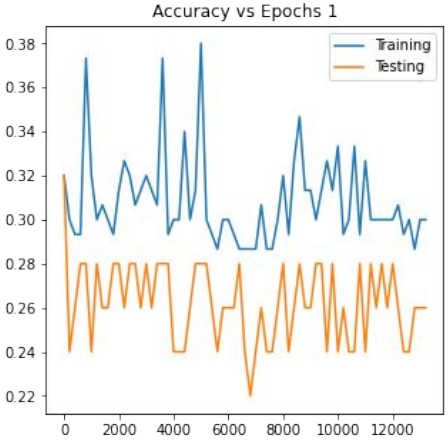
A la hora de evaluar la capacidad de generalización del perceptrón, decidimos seguir utilizando los parámetros con los que menor error encontramos, ya que aunque los pesos que se creasen a partir de los diferentes subconjuntos de entrenamiento serán diferentes, el beta, la función sigmoide y la tasa de aprendizaje siguen siendo parte del perceptron a evaluar.

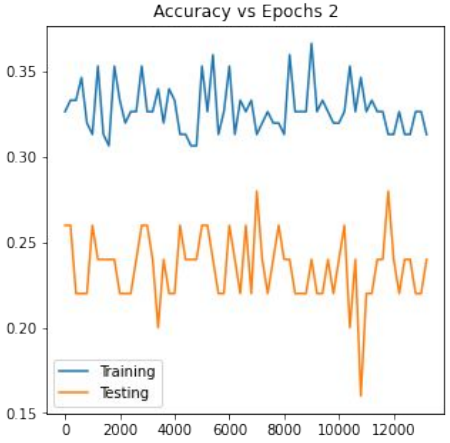
```
public double ACCURACY_EPSILON = 0.18;
```

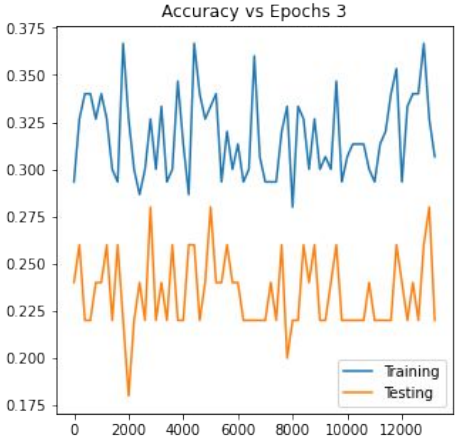
```
double testAcc = calculateAccuracy(w, testingSetInputs, testingSetOutputs, ACCURACY_EPSILON);
```

Con k = 4

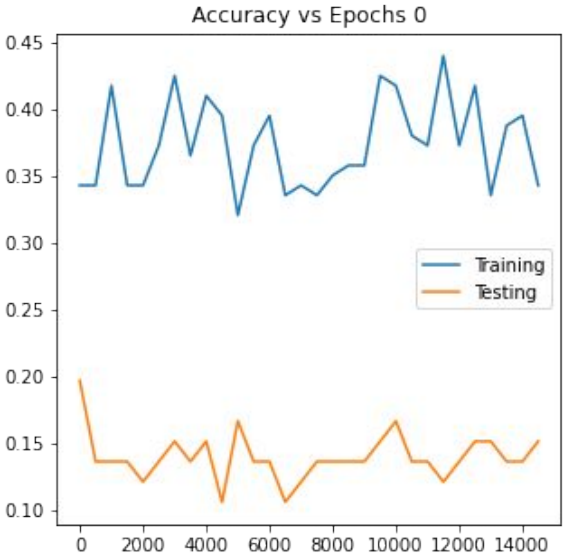

Accuracy vs Epochs 0
Training
Testing
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
2000
4000
6000
8000
10000
12000

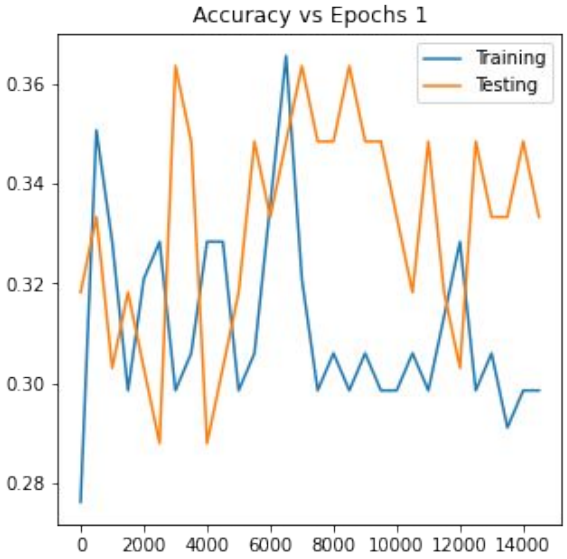

Accuracy vs Epochs 1
Training
Testing
0.38
0.36
0.34
0.32
0.30
0.28
0.26
0.24
0.22
0
2000
4000
6000
8000
10000
12000

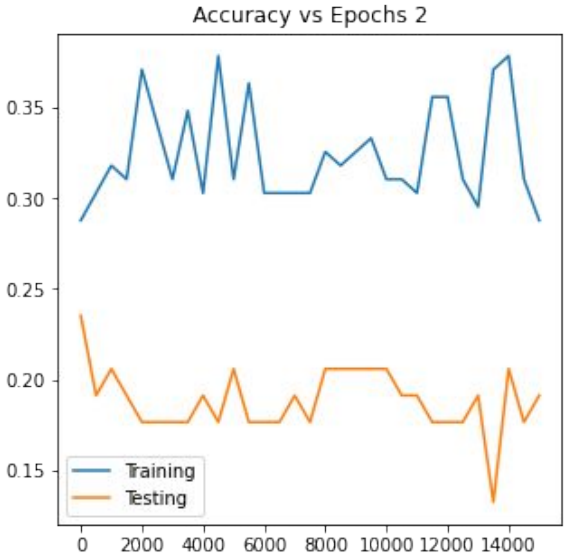

Accuracy vs Epochs 2
Training
Testing
0.35
0.30
0.25
0.20
0.15
0
2000
4000
6000
8000
10000
12000


Accuracy vs Epochs 3
Training
Testing
0.375
0.350
0.325
0.300
0.275
0.250
0.225
0.200
0.175
0
2000
4000
6000
8000
10000
12000

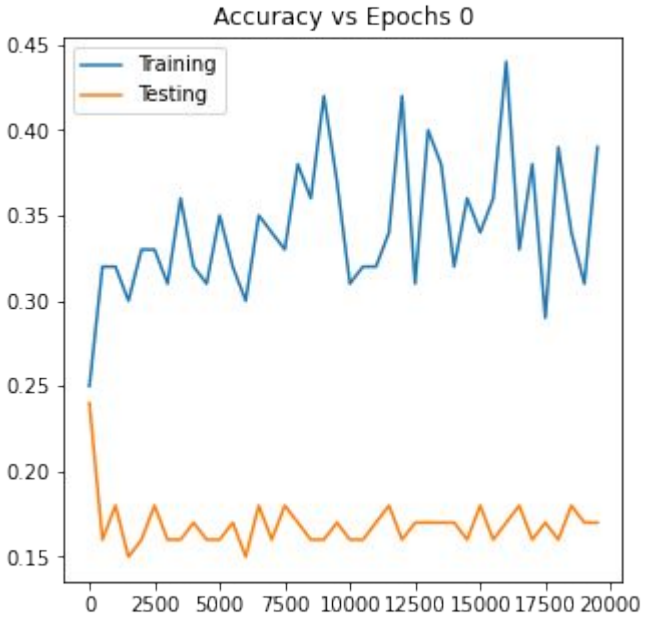
Con k = 3

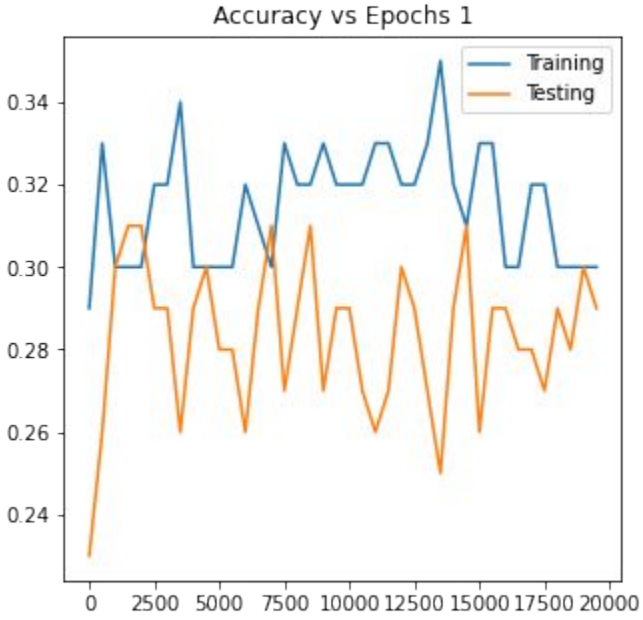

Accuracy vs Epochs 0
Training
Testing
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
2000
4000
6000
8000
10000
12000
14000


Accuracy vs Epochs 1
Training
Testing
0.36
0.34
0.32
0.30
0.28
0
2000
4000
6000
8000
10000
12000
14000


Accuracy vs Epochs 2
Training
Testing
0.35
0.30
0.25
0.20
0.15
0
2000
4000
6000
8000
10000
12000
14000

Con k = 2


Accuracy vs Epochs 0
Training
Testing
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0
2500
5000
7500
10000
12500
15000
17500
20000


Accuracy vs Epochs 1
Training
Testing
0.34
0.32
0.30
0.28
0.26
0.24
0
2500
5000
7500
10000
12500
15000
17500
20000

Sobre el perceptrón multicapa

Para el ejercicio 3 se nos dio un conjunto de datasets de entradas y salidas. La implementación del perceptrón multicapa sigue la siguiente arquitectura:

- El perceptrón es un arreglo de capas
- Cada capa tiene un arreglo de unidades
- Cada unidad calcula y conoce su excitación, activación y pesos sinápticos

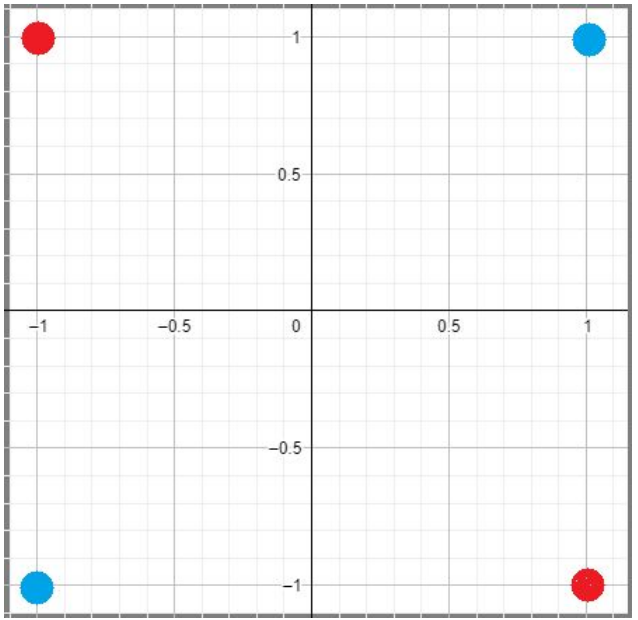
Ejercicios del perceptrón multicapa

Ejercicio 3.1: dada la función XOR calcular la salida

Entrada: x = {{−1, 1}, {1, −1}, {−1, −1}, {1, 1}}

Salida esperada: y = {1, 1, −1, −1}

Al no ser un problema linealmente separable, debe utilizarse un perceptrón multicapa



Ejercicios del perceptrón multicapa

Ejercicio 3.1: dada la función XOR calcular la salida

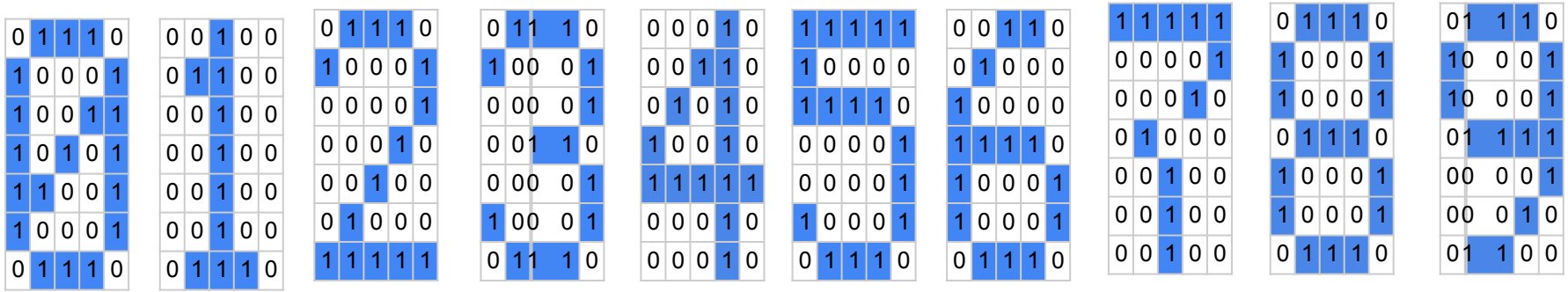
Se calculó la matriz de confusión: esperado vs resultado

		Resul	tado
		1	-1
Esperado	1	22	0
	-1	1	24

Epochs	500000
Corridas	12
beta	1
Learning rate	0.01
Capas ocultas	{4, 3}

Ejercicios del perceptrón multicapa

Ejercicio 3.2: dados píxeles que forman números, definir la paridad



Entrenar con un subconjunto y testear con otro

Gráfico de accuracy

Acc= bien clasificados/total

Epochs	100001
beta	1
Learning rate	0.01
Capas ocultas	{25, 20, 15, 10, 5}
Accuracy error	0.18

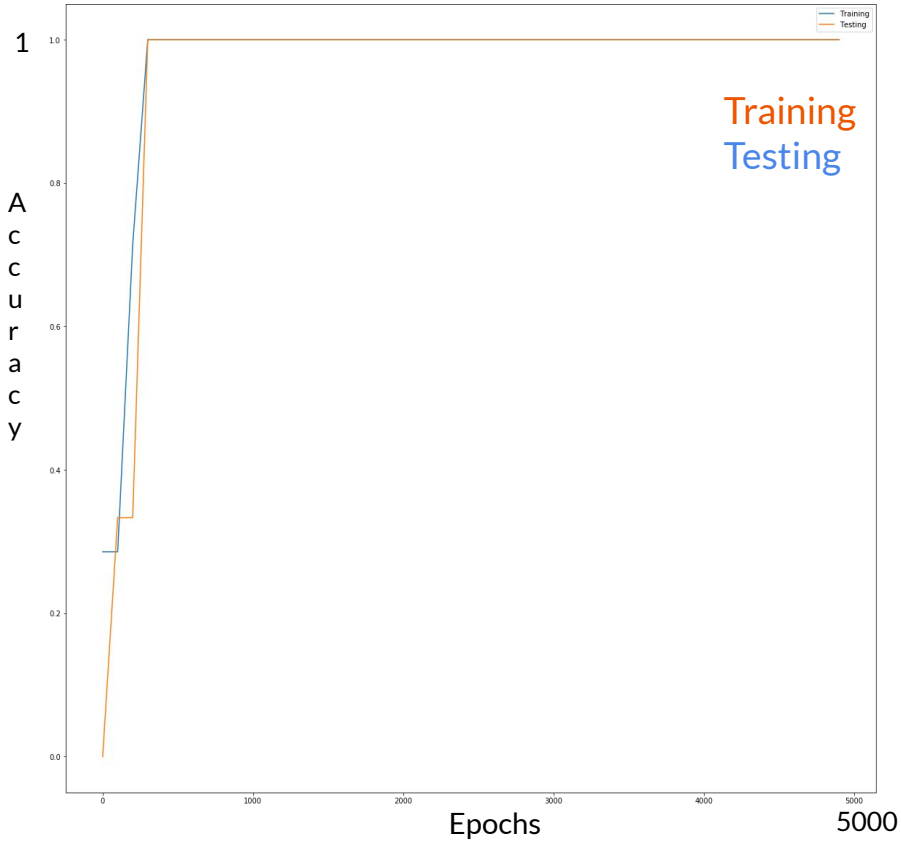
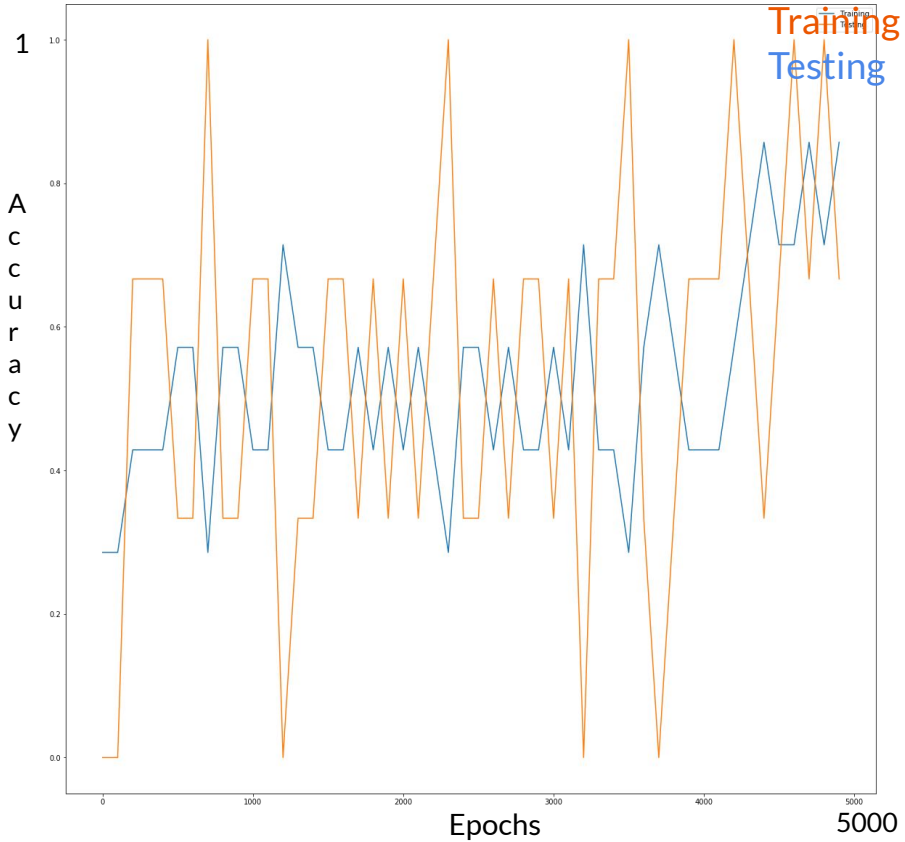


Gráfico de accuracy

Acc= bien clasificados/total

Epochs	5000
beta	1
Learning rate	0.01
Capas ocultas	{25, 20, 15, 10, 5}
Accuracy error	0.1



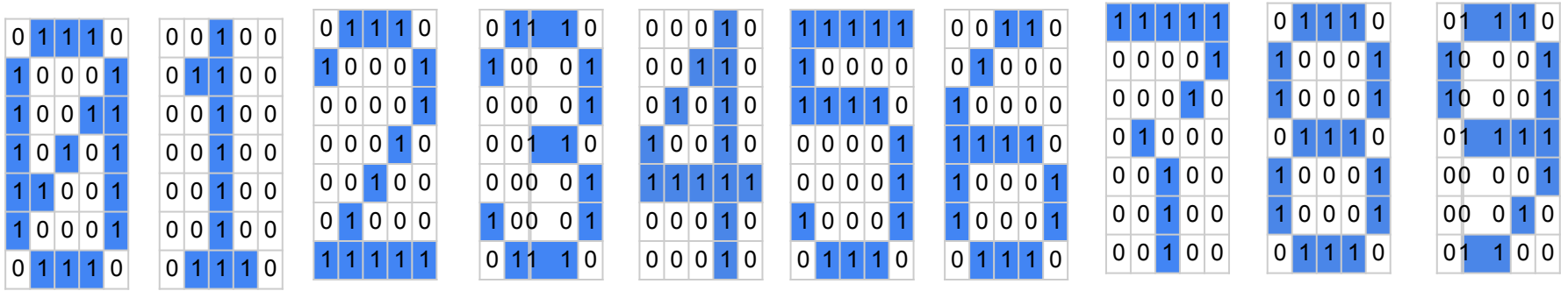
Ejercicios del perceptrón multicapa

Ejercicio 3.2: dados píxeles que forman números, definir la paridad

¿Qué podría decir acerca de la capacidad para generalizar de la red?

Ejercicios del perceptrón multicapa

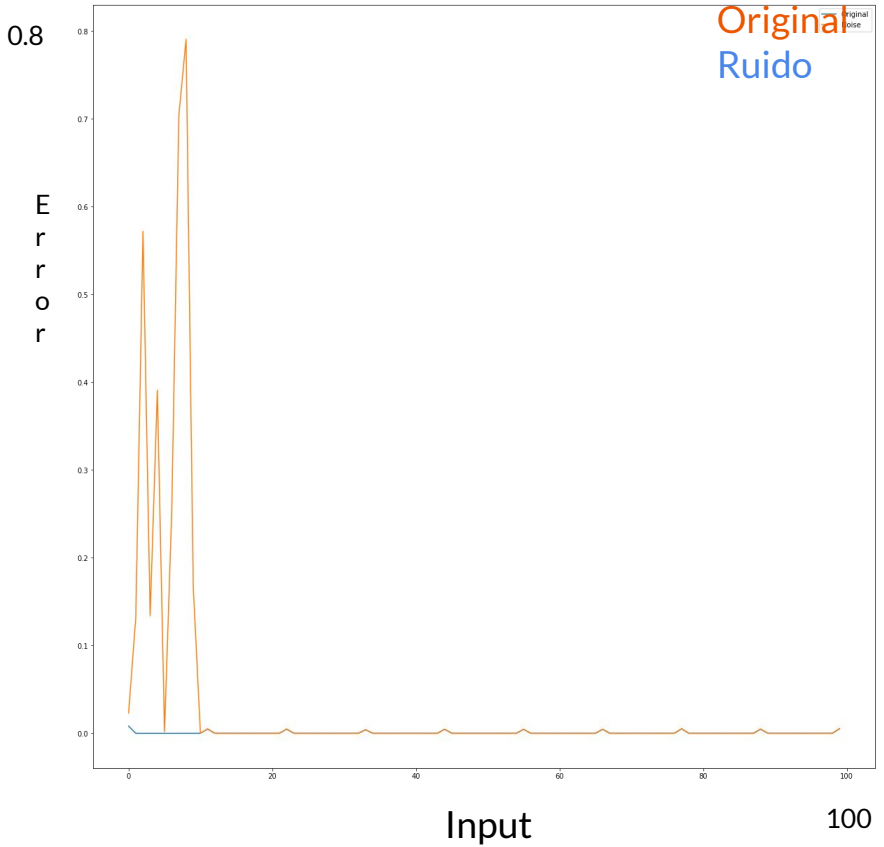
Ejercicio 3.3: dados píxeles que forman números, prender el bit de salida correspondiente



Una vez que la red aprendió, aplicar ruido

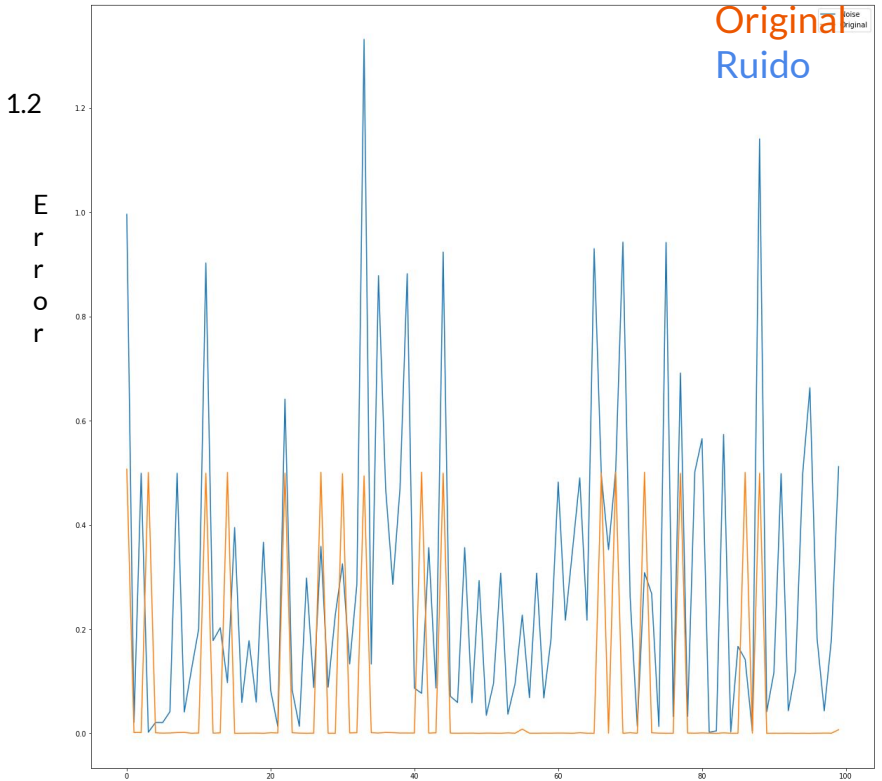
Error del ruido

Epochs	100001
beta	1
Learning rate	0.01
Capas ocultas	{25, 20, 15, 10, 5}
Probabilidad ruido	0.02



Error del ruido

Epochs	100001
beta	1
Learning rate	0.01
Capas ocultas	{25, 20, 15, 10, 5}
Probabilidad ruido	0.5



Muchas gracias

Grupo 19
Integrantes:

- ➔ Lucas Catolino
- ➔ Matias Ricarte