Perceptrón simple y multicapa
SIA-TP3
2022 1C

Sicardi, Julián Nicolás - Legajo 60347

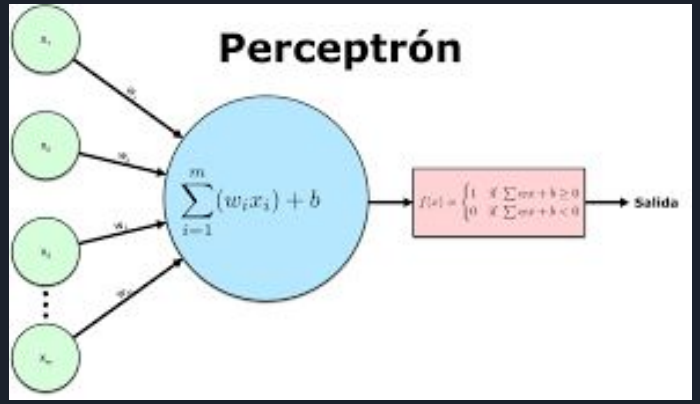
Quintairos, Juan Ignacio - Legajo 59715

Zavalía Pángaro, Salustiano J. - Legajo 60312

Introducción

Descripción del proyecto

- Implementación de algoritmo del perceptrón simple (escalón, lineal, no lineal).
- Implementación de algoritmo del perceptrón multicapa.
- Análisis de resolución de problemas con distintos tipos de perceptrones y parámetros.

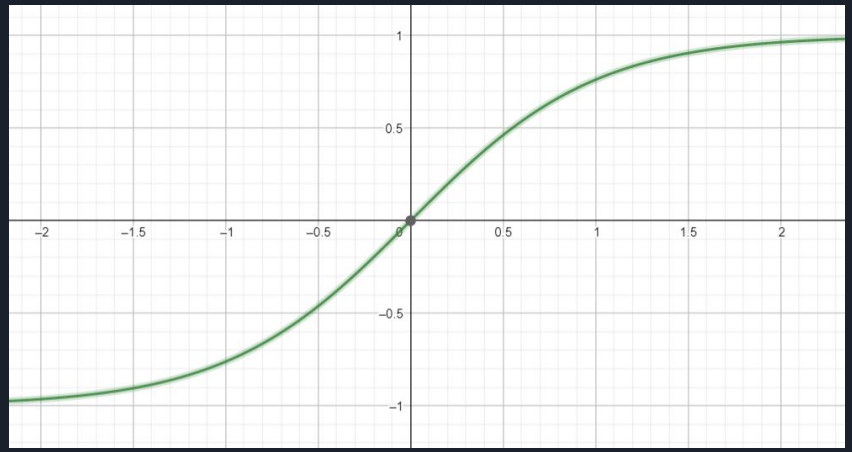


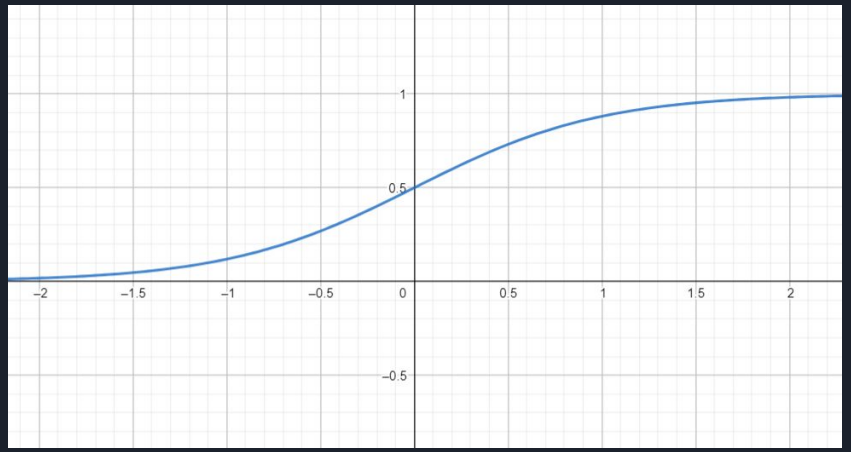
Perceptrón simple escalón

- Función de activación binaria (tomamos los valores -1 y 1)
- Objetivo es resolución de problemas:
  - AND lógico.
  - XOR lógico.
- Entrada:
  - Combinaciones de dos elementos (-1 y 1).
- Salida:
  - Estimación de la función lógica correspondiente.

Perceptron lineal y no lineal

- Perceptron lineal:
  - Funcion de activacion: O(h) = h.
- Perceptrón no lineal:
  - Funciones de activación: O(h) = tanh(βh) y O(h) = 1 / (1 + e^-2βh).



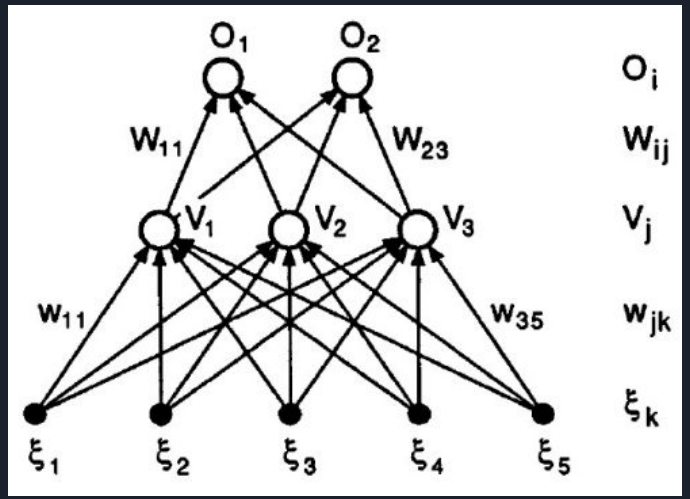


Perceptrón lineal y no lineal

- Entrada: vectores de 3 componentes reales.
- Salida: valor real por cada input.
- Objetivo: aproximación de valores de salida.
- Análisis de generalización de perceptrón simple no lineal.
- ¿Cómo podría escoger el mejor conjunto de entrenamiento?.
- ¿Cómo podría evaluar la máxima capacidad de generalización?.

Perceptrón multicapa

- Para resolver problemas más complejos se trabaja con capas de neuronas.



- Cada neurona recibe los valores de activación de la capa inferior y emite hacia arriba.
- Se propaga el error hacia abajo para luego actualizar pesos.

Perceptrón multicapa

- Problema XOR.
- Números pares:
  - Entrada: píxeles de imagen de 5x7 que forma el número.
  - Salida: 1 si es par, 0 si es impar.
- Análisis de generalización de perceptrón.
- Números:
  - Salida: vector de 10 valores (1 si es numero, 0 sino).
- Análisis con probabilidad de ruido en pixeles.
- Softmax.

Implementación

Configuración inicial

- Archivo config.json.

```json
{
    "perceptron_type" : "multilayer",
    "hidden_layers": [2],
    "entry_file": "resources/training_ej3.txt",
    "output_file": "resources/zeta_linear.txt",
    "problem": "XOR",
    "learning_rate" : 0.1,
    "max_epochs" : 500,
    "min_error": 1e-10,
    "sigmoid_type" : "tanh",
    "beta": 0.5,
    "alpha": 0,
    "cross_validate":"False",
    "test_proportion": 0.2,
    "softmax": "False"
}
```

Configuración inicial: Parámetros

Los parámetros son de GRAN importancia!

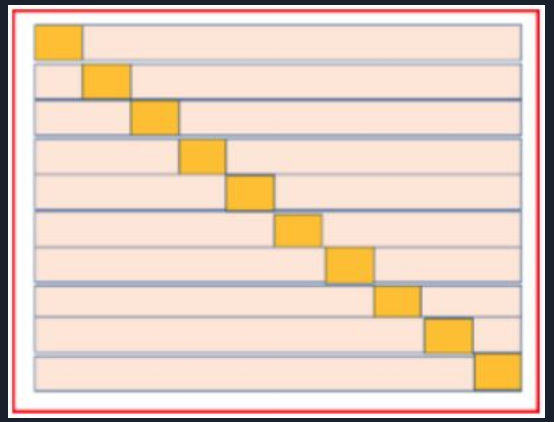
- Tipo de perceptrón: perceptrón puede no poder resolver el problema.
- Capas ocultas (multicapa) : Se suelen necesitar más neuronas/capas cuanto más complejo el problema.
- Tasa de aprendizaje : Valores bajos hacen que tarde en converger. Valores altos puede que no converja.
- Épocas : Un valor bajo no da tiempo de aprender lo suficiente.
- Beta (no lineal y multicapa) : Afecta el comportamiento de la función de activación.
- Alpha (multicapa) : Qué tanto afecta el cambio en el peso del paso anterior al actual.

Diseño multicapa

- Objetos para capas, neuronas y neuronas umbral.
- Cada capa contiene neuronas.
- Cada neurona contiene los pesos que van de ella a la capa inferior.
- Las neuronas umbral (presentes en cada capa excepto en la de salida) siempre devuelven la misma activación y no tienen pesos hacia abajo.

Validación cruzada

- Tomamos el parámetro “`test_proportion`” de la configuración y conseguimos el k haciendo: *k = floor(1 / test.proportion)*
- Segmentamos los conjuntos de entrada y salida esperada en *k* partes
- Ciclamos por los *k*, haciendo que el k-ésimo segmento de ambos conjuntos sea usado para las pruebas y el resto para entrenamiento



Métricas: perceptrón lineal y no lineal

- Para el perceptrón simple lineal y no lineal definimos una “*accuracy*” alternativa como:

  accuracy = hits / total_values

  Siendo *hits* la cantidad de entradas donde:

  |calculated_value - expected_value| < error_threshold

  total_values la cantidad de valores del conjunto de salida

  y *error_threshold* un umbral de tolerancia

Métricas: Perceptrón Multicapa

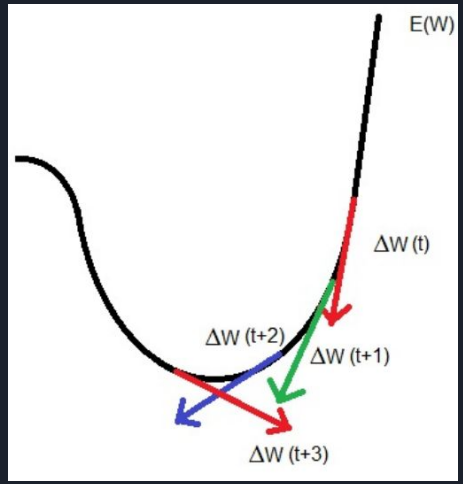
- Utilizamos la matriz de confusión para implementar las métricas:
  - *accuracy*
  - *precision*
  - *recall*
  - *f1_score*
  - *true_positives_rate*
  - *false_positives_rate*

  Como se vieron en la clase teórica

Actualización normal vs Momentum

- Con momentum, los valores de actualización de los pesos toman también una proporción de la actualización anterior.
- Ayuda a escapar de zonas de valle en las cuales el algoritmo normal se estancaria. Y ayuda a acelerar el cambio en zonas planas.

$$\Delta w_{ij}(t+1) = -\eta \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w_{ij}(t)$$



Resultados

Objetivo y Gráficos

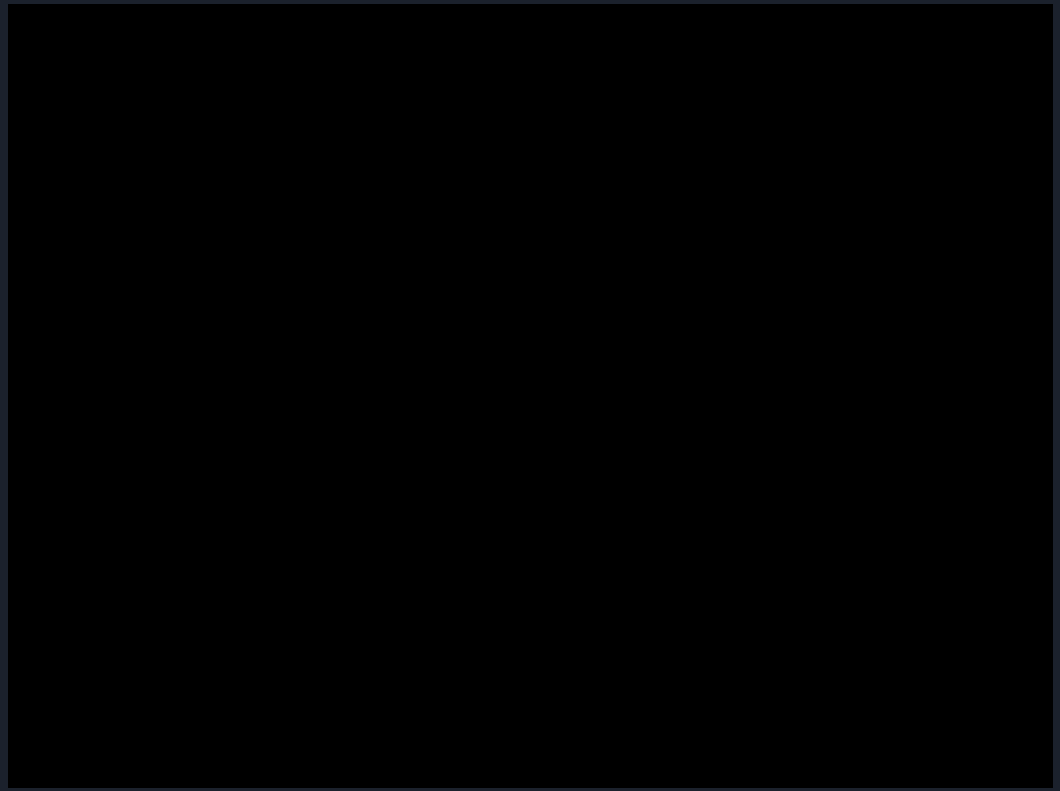
- Objetivo: Estudiar cómo varían los resultados en base a los parámetros y ver qué problemas puede resolver cada perceptrón
- Observables: ObservableHQ
    - Errores (entrenamiento y/o prueba) vs parámetro
    - Visualización puntos a clasificar e hiperplano generado por la red

Ejercicio 1

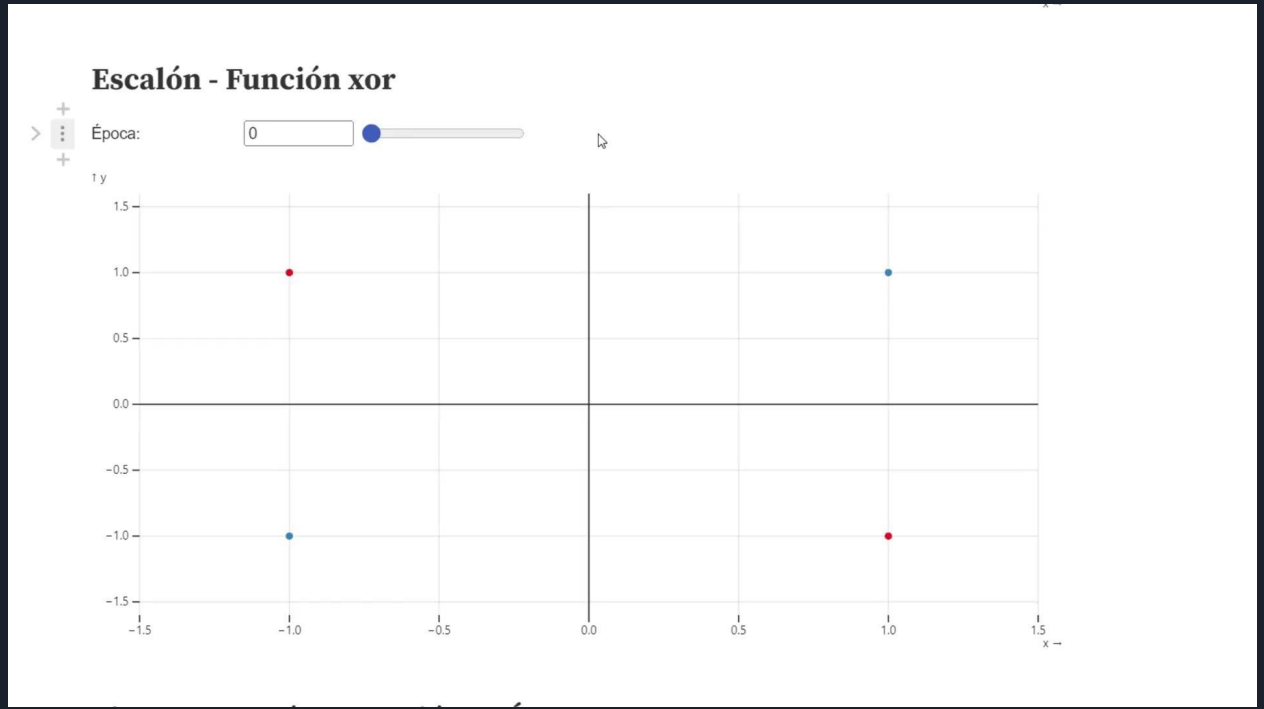
Objetivo

- Evaluar la capacidad del perceptrón simple escalón de resolver los problemas:
  - AND.
  - XOR.
- Visualizar la recta de separación generada

Escalón - AND



Escalón - XOR


Escalón - Función xor
Época:
0
y
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x

Análisis: Perceptrón Simple Escalón

- Como el perceptrón simple escalón resuelve problemas de separabilidad lineal:
  - Resuelve correctamente el AND.
  - No resuelve el XOR:
    - Esto ocurre porque no existe una única línea que separe las dos regiones.

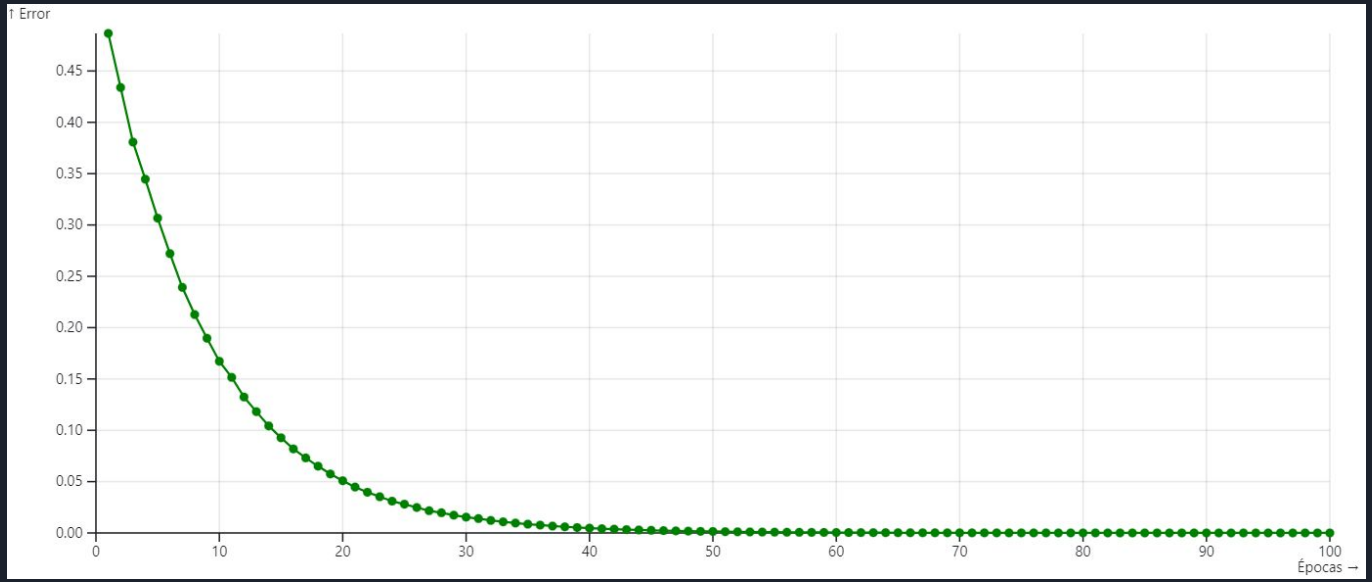
Ejercicio 2

Objetivo

- Evaluar la capacidad del perceptrón simple lineal y no lineal para resolver el problema de aproximación:
  - Variar parámetro beta para el perceptrón no lineal.
  - Variar la función de activación en el perceptrón no lineal.
- Evaluar la capacidad de generalización del perceptrón simple lineal y no lineal
  - Empleo de validación cruzada.
- Evaluar la elección del mejor conjunto de entrenamiento
  - Variar la proporción del conjunto de prueba durante la validación cruzada.

Perceptrón simple lineal:

Para un problema lineal inventado:

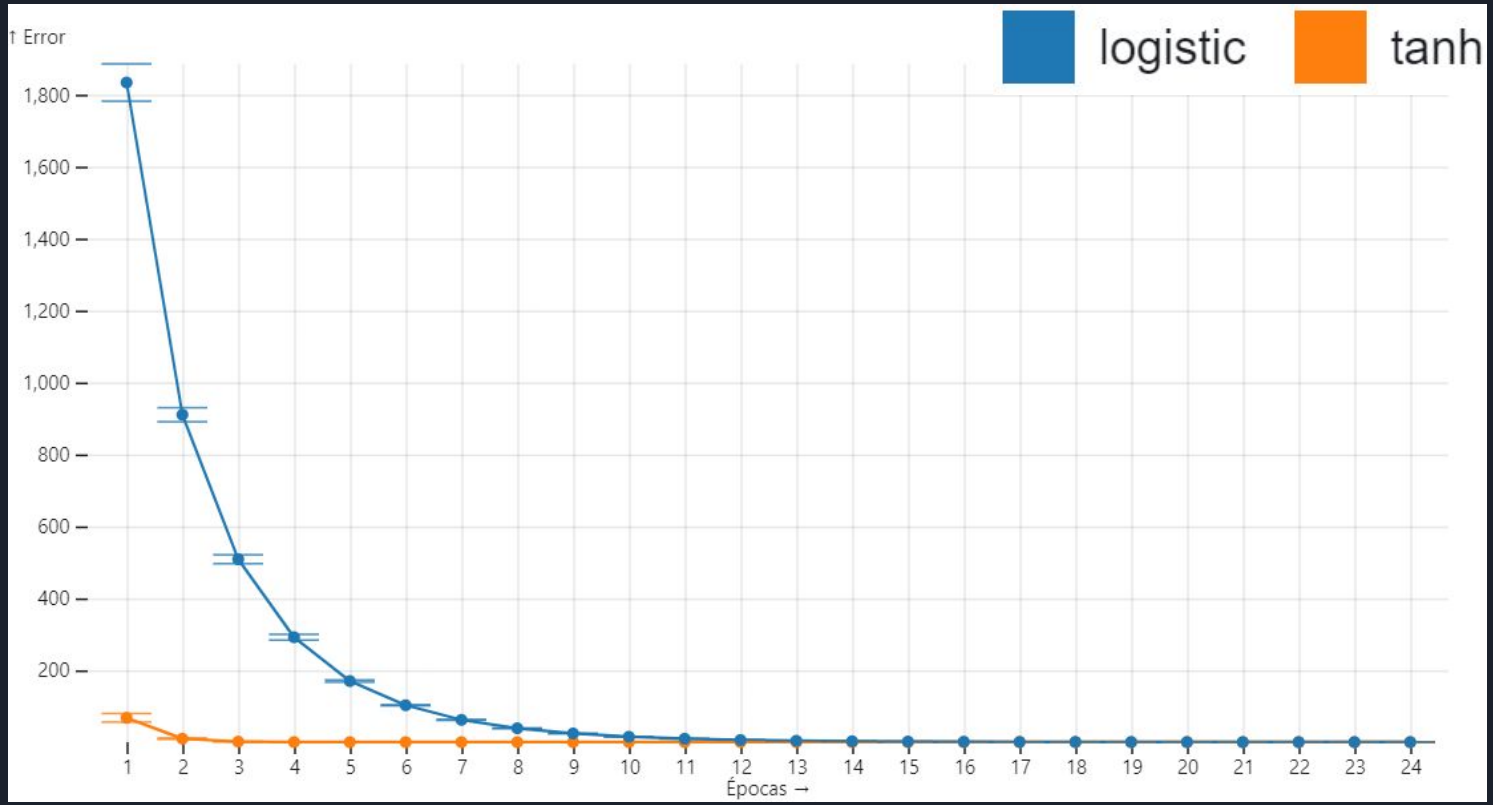


Para el problema del ej2:

Errores constantes del perceptrón lineal en sus 3 corridas: 296860, 1598563, 1256766

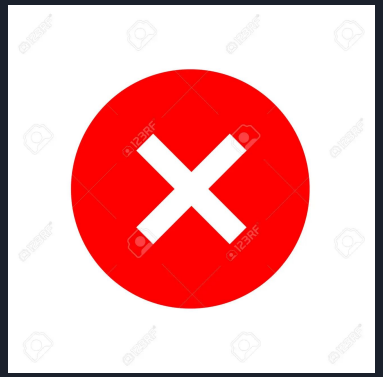
Ingraficable! No puede resolver el problema dado

Perceptrón simple no lineal: activación



Explicación resultados ejercicio 2

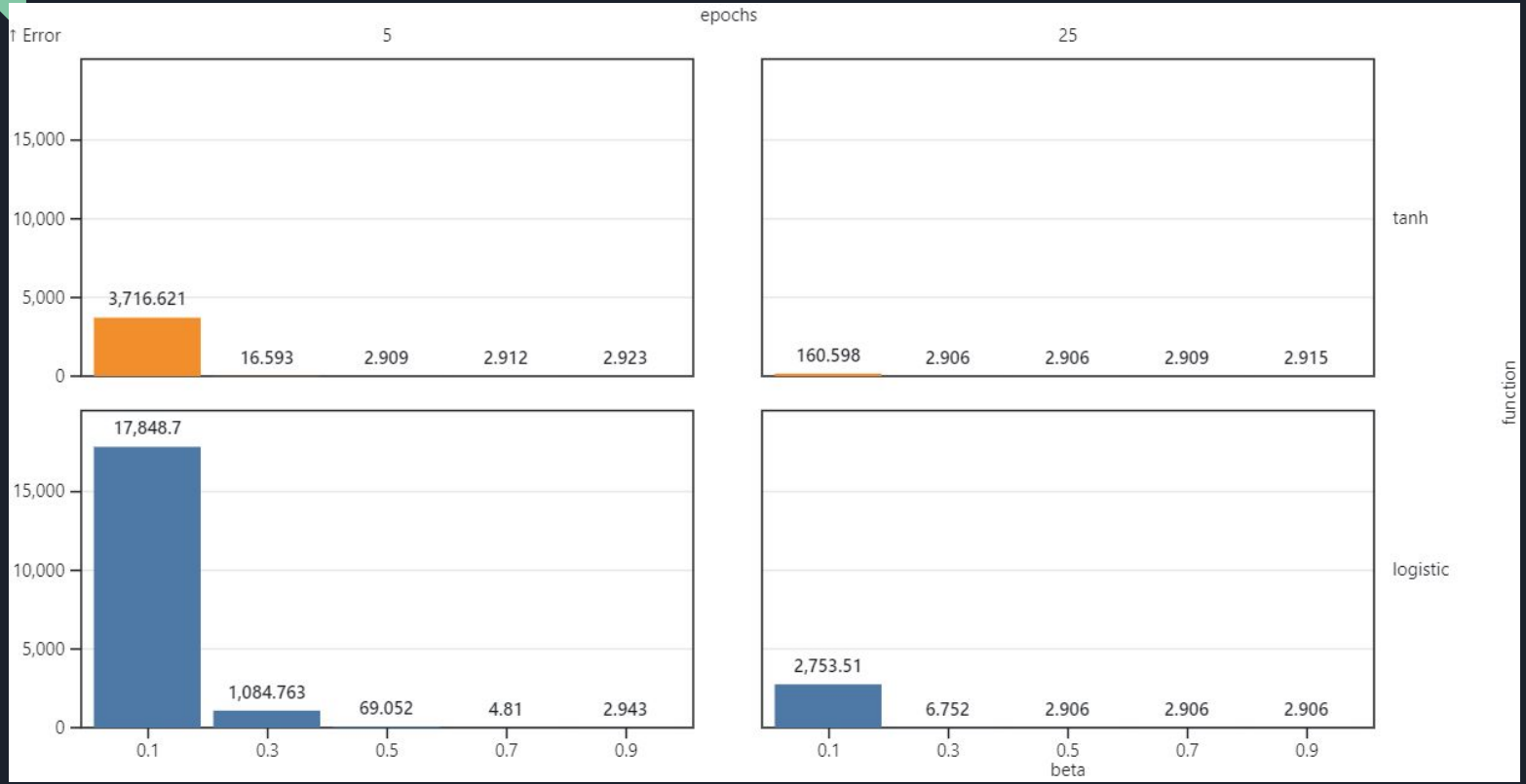
- El perceptrón simple lineal tiene grandes dificultades porque el problema no es de naturaleza lineal.
  - No logra aprender durante el entrenamiento.
- El perceptrón simple no lineal puede resolver el problema.
  - El aprendizaje varía con los parámetros utilizados



Análisis: Funciones de activación

- En este caso la función tanh comienza arrojando mejores resultados y es por eso que llega rápidamente a valores bajos de error
- Aunque la logística devuelve errores más altos, luego de varias épocas llega a un error del mismo orden que la función tanh

Perceptrón simple no lineal: Beta óptimo



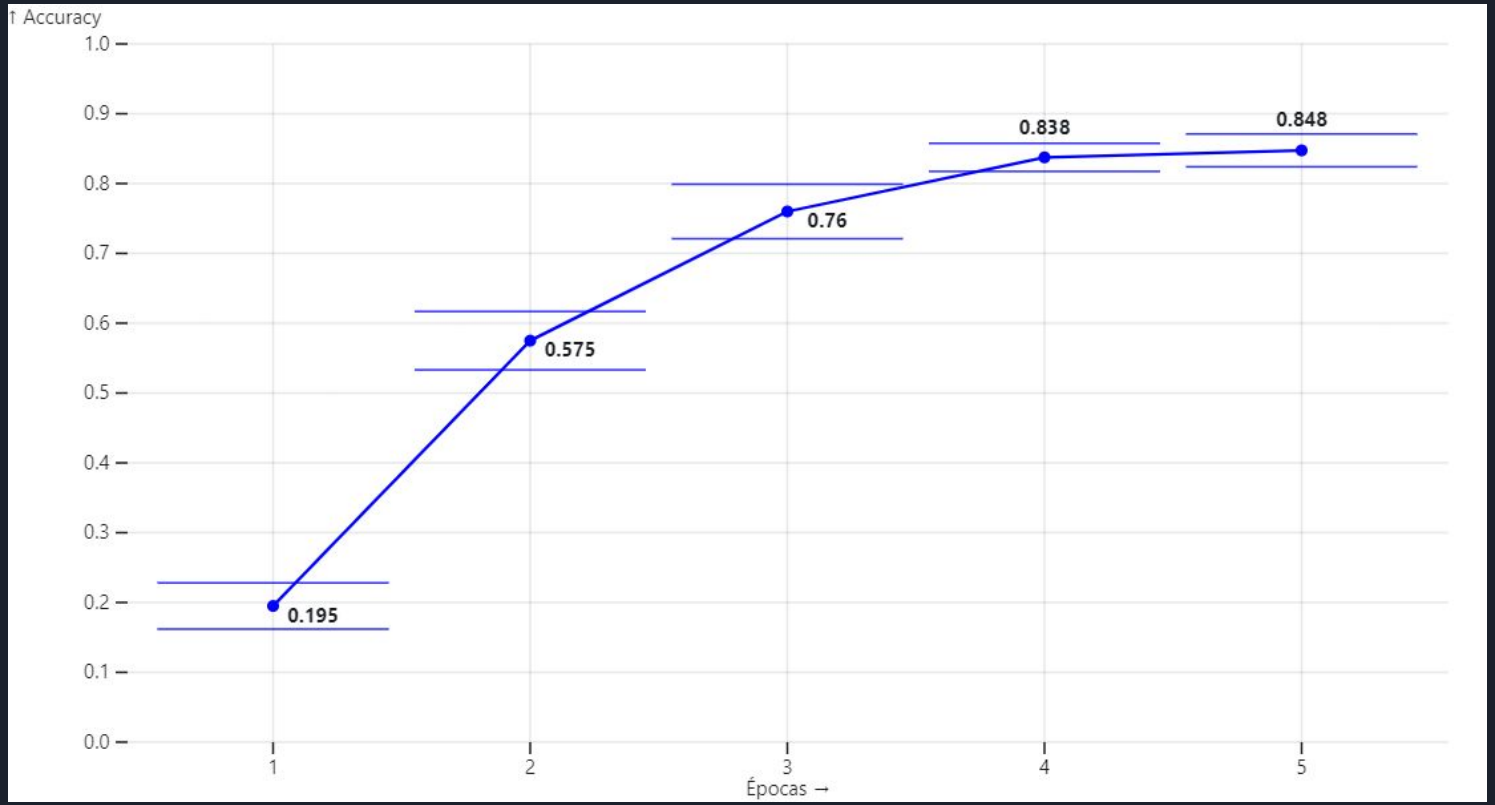
Análisis: Beta

- El valor de beta hace que la función sigmoidea de activación se comporte más parecido a una lineal o una escalón.
- En nuestro caso, valores altos de beta arrojan mejores resultados. Hacen que la red aprenda más rápido
- Con beta = 0.1 la función logística alcanza error ≈ 2.905 en 1000 épocas!

Generalización ejercicio 2

- Se hace uso de la validación cruzada para intentar escoger el mejor conjunto de entrenamiento y evaluar su capacidad de generalización.
- Probamos distintas divisiones y estudiamos para cuáles se arrojan los mejores resultados usando las métricas de evaluación
- Calculamos el *accuracy* como se explicó anteriormente, con un *error_threshold* de 0.5.
- Variación de épocas máximas para ver cómo varía *accuracy*

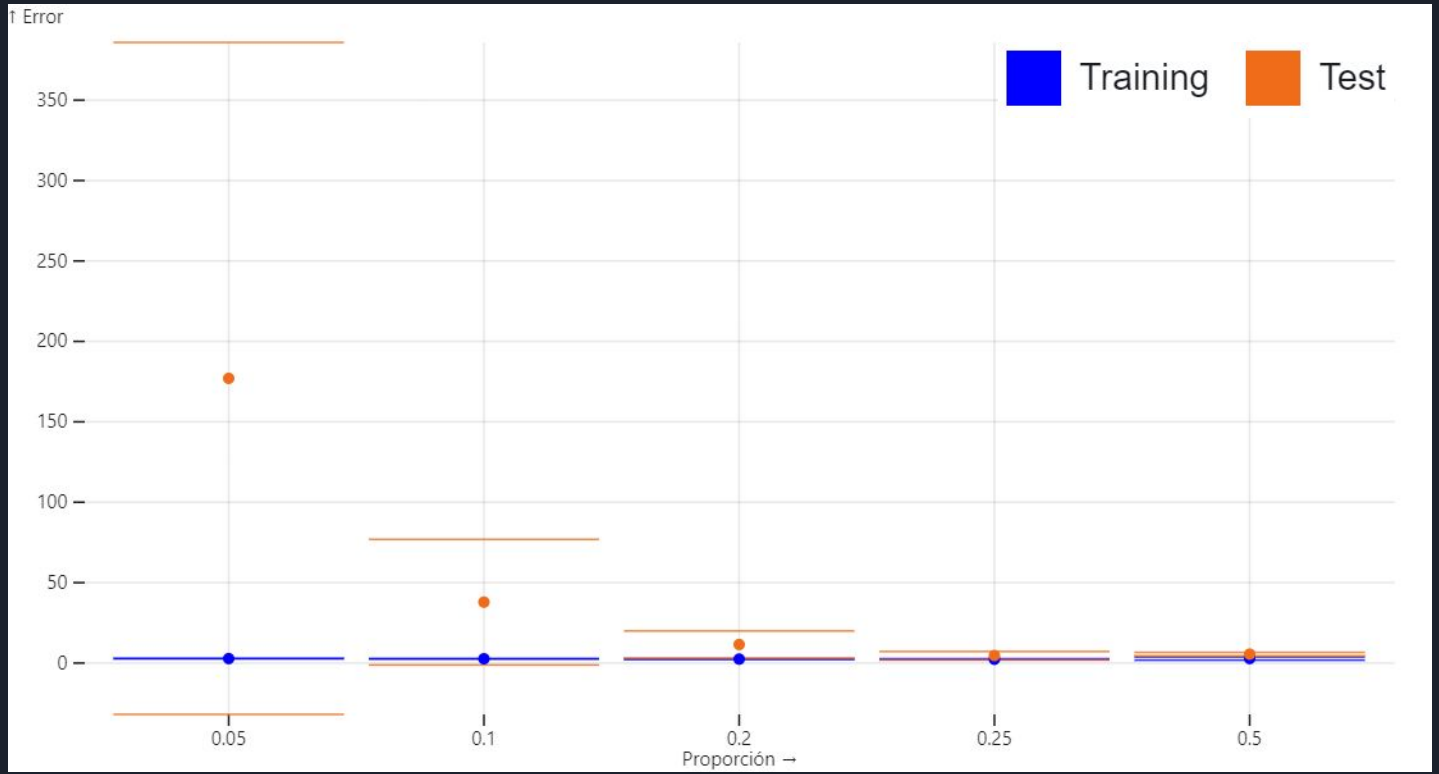
Épocas vs *accuracy*



Análisis: Accuracy

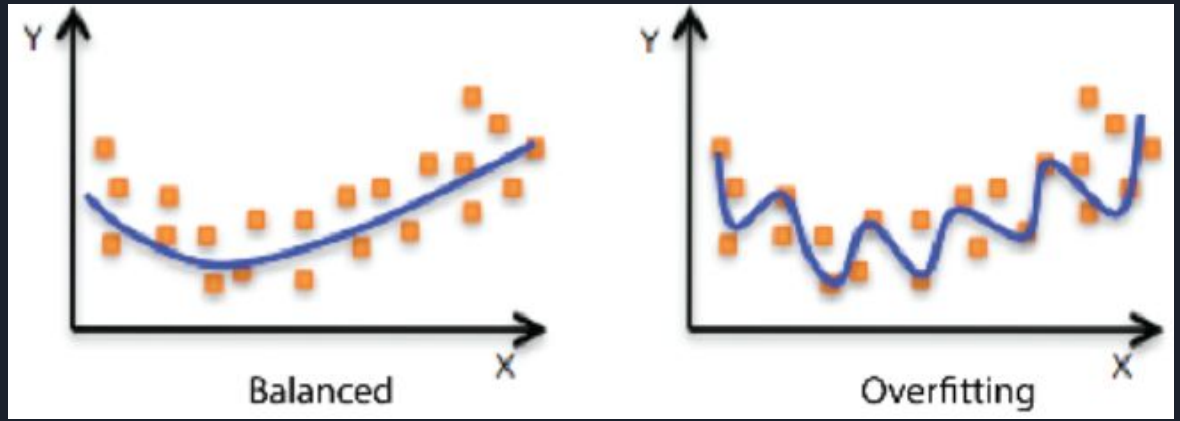
- Se observa que a mayor cantidad de épocas la red aprende más y en consecuencia la accuracy aumenta.
- Sin embargo, se tiene un comportamiento asintótico y la *accuracy* nunca llegará a 1.

Proporción de prueba óptima



Análisis: Proporción de prueba

- Si se dejan pocas entradas para las pruebas se observa un comportamiento de sobreajuste.
- Pero si se toman demasiadas para pruebas los resultados tampoco son buenos

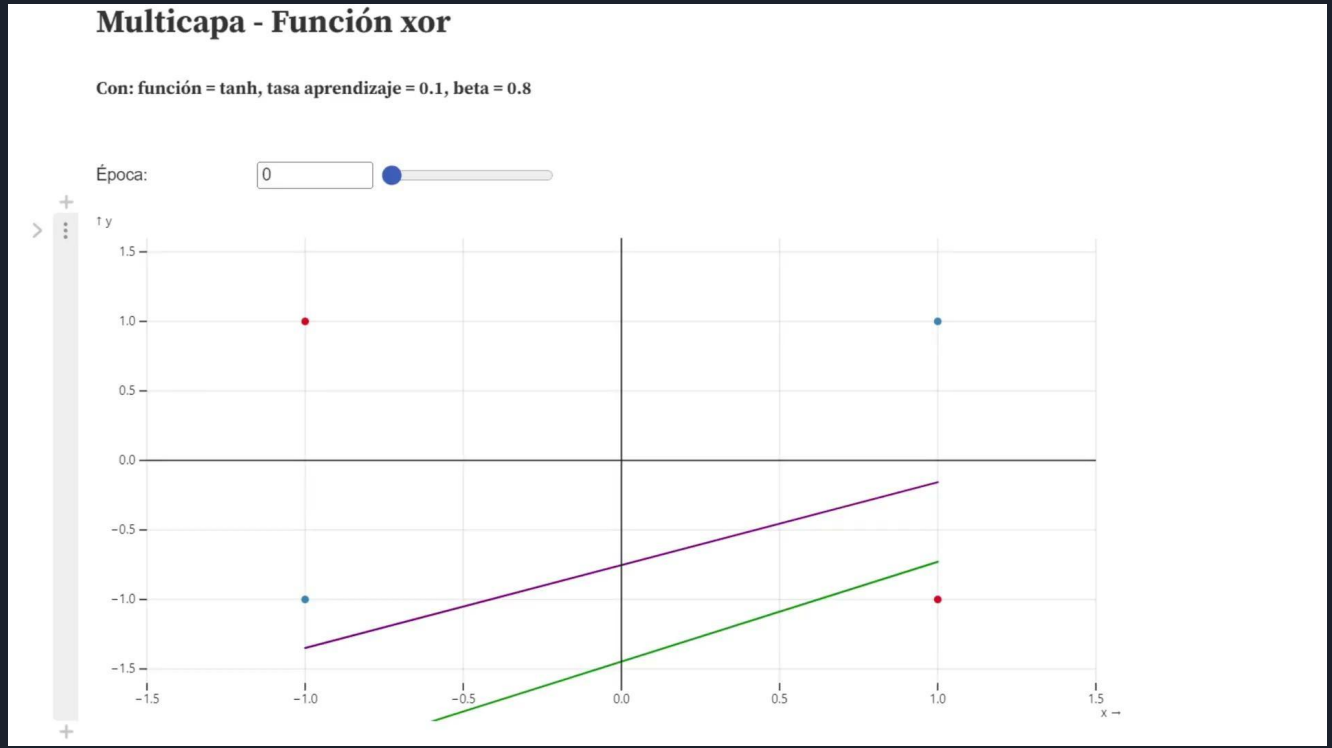


Ejercicio 3

Objetivos

- Evaluar la capacidad del perceptrón multicapa de resolver el problema del XOR.
- Evaluar la capacidad de generalización de la determinación de si un número es par:
  - Utilizar validación cruzada para tomar un subconjunto de los dígitos como conjunto de prueba.
- Evaluar la capacidad de generalización de la identificación de un dígito:
  - Utilizar los dígitos originales para entrenar.
  - Utilizar los dígitos afectados por ruido para las pruebas.

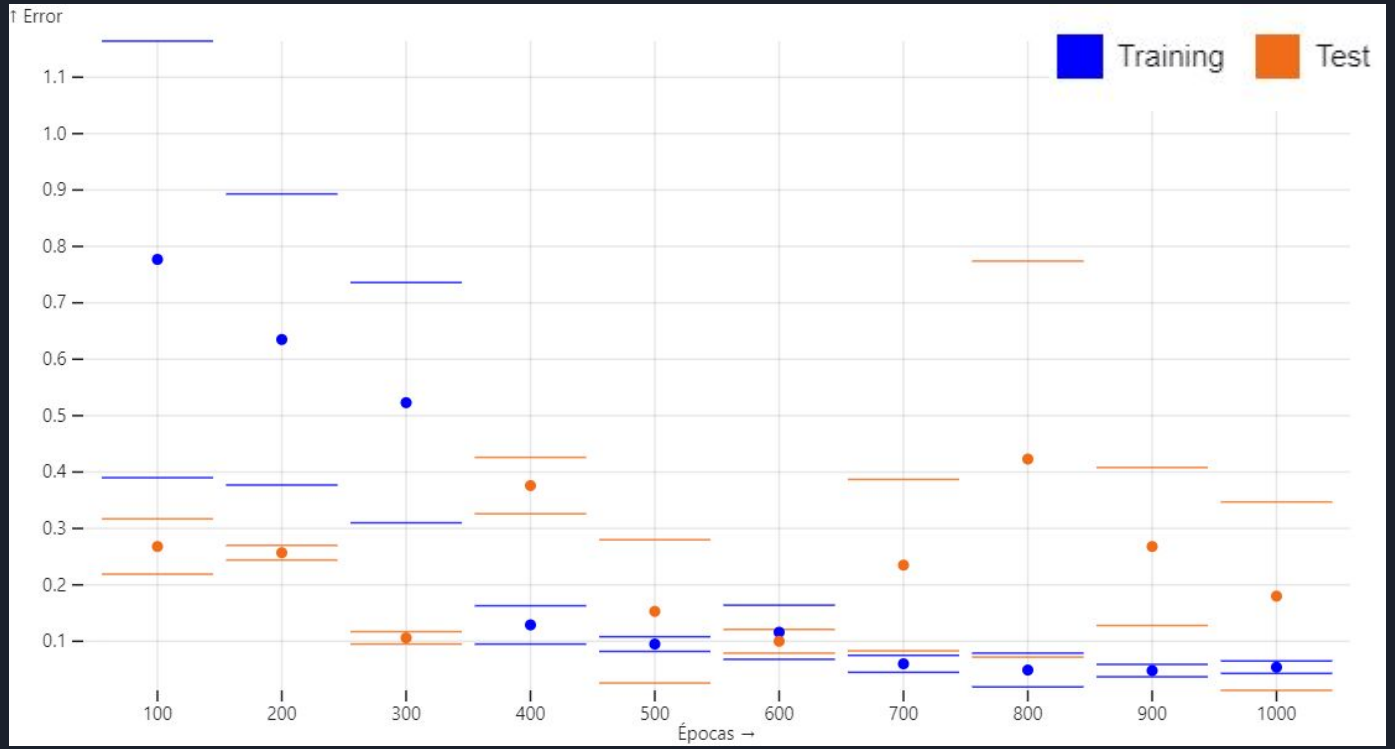
Multicapa - XOR



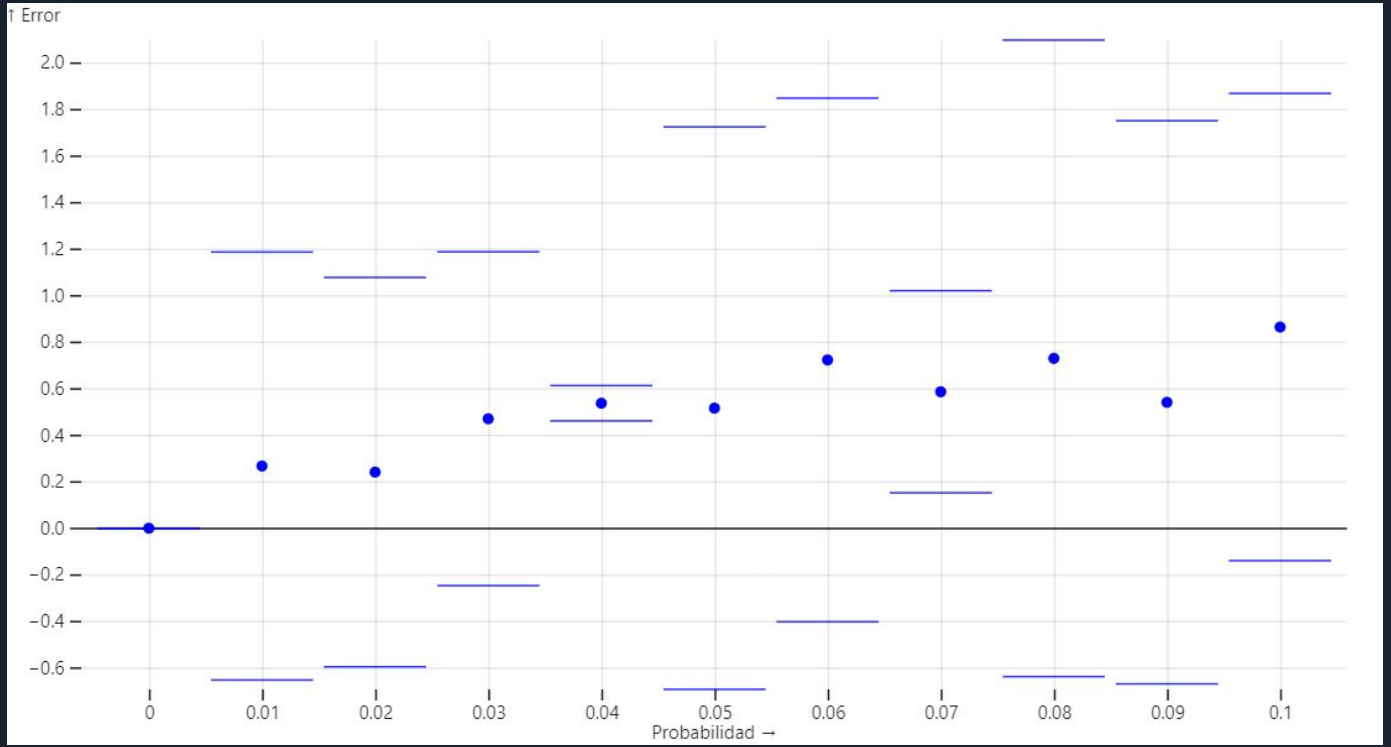
Análisis: XOR multicapa

- A diferencia del perceptrón simple, el perceptrón multicapa puede resolver el problema del XOR
- Esto es porque el perceptrón multicapa puede resolver problemas que no son linealmente separables.
- En este caso se puede pensar como una combinación de perceptrones simples y por eso se puede representar con varias rectas en un plano

Multicapa - Pares sin ruido: Error *vs.* épocas



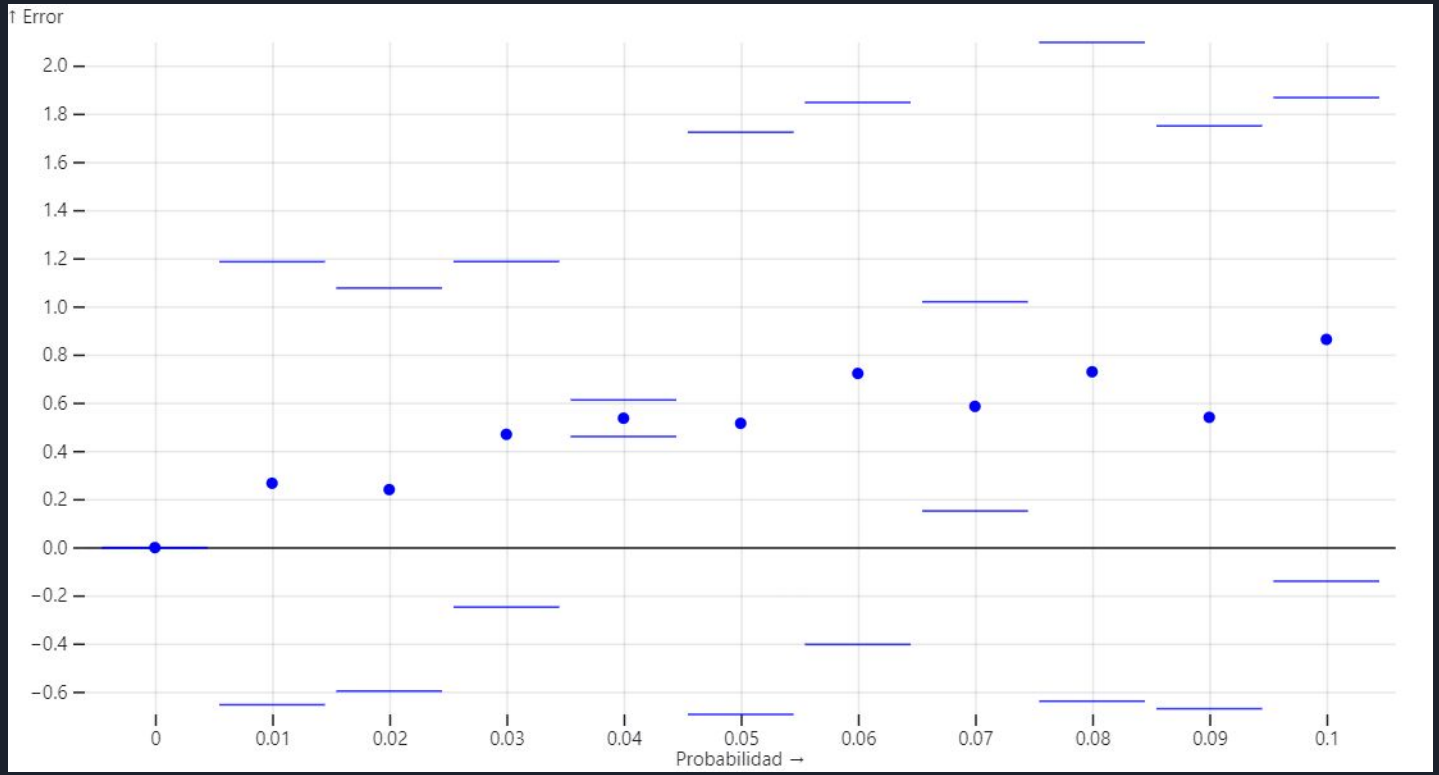
Multicapa - Pares: Prob. de ruido *vs.* error



Análisis: Pares

- La capacidad de generalización del perceptrón para este problema es mala.
- Conjunto de prueba contiene dígitos que nunca ha visto y éstos no son lo suficientemente parecidos a los demás de su clase, dificultando mucho que se arroje una respuesta acertada.
- Si entrenamos con el conjunto de números original y lo probamos con ruido tenemos mayor capacidad de generalización.

Multicapa - Dígitos: Prob. de ruido *vs.* error



Análisis: Dígitos

- La capacidad de identificar correctamente un número del perceptrón aumenta de forma bastante constante en relación a la probabilidad de ruido.
- Se observa un desvío estándar grande porque el rendimiento varía considerablemente en relación a la cantidad y posición del ruido, que es completamente probabilístico.

Activación normal vs softmax

- Softmax mapea los valores de un vector a probabilidades

$$\sigma\left(\begin{bmatrix}1.2\\0.3\end{bmatrix}\right)$$

$$\begin{bmatrix}\frac{e^{1.2}}{e^{1.2}+e^{0.3}}\\ \frac{e^{0.3}}{e^{1.2}+e^{0.3}}\end{bmatrix}$$

$$\begin{bmatrix}0.71\\0.29\end{bmatrix}$$

Esperando: [0, 0, 0, 0, 0, 0, 0, 0, 1]

Normal:

```
Activations: [0.009535596562262159, 0.0003673095498855519, 0.0016159548912248749, -0.018145664476537184
, 0.0021419836530816943, -0.009181179907279506, 0.004356032187289404, -0.015549783319708093, -0.0102695
54512528443, 0.9724980350320832]
```

Con softmax:

```
Activations: [0.06105487814805697, 0.06098778584359358, 0.061305495066896004, 0.061083619683205144, 0.0
6098766501407475, 0.06121924690481541, 0.06128148536759856, 0.06179479608444338, 0.06105603652797165, 0
.4492289913593444]
```

Justo para este problema no está muy bueno de aplicar porque e^0.00x ≈ 1. Útil si los valores son superiores a 1.

Conclusiones

Conclusiones

- En varios casos los perceptrones no resuelven todos los problemas requeridos o su desempeño sufre.

- El comportamiento de la red varía considerablemente en función de los parámetros de configuración.

- Error bajo de entrenamiento no implica error bajo de testeo. No siempre se generaliza bien.

¡Muchas Gracias!