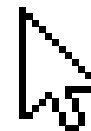


(72.27) Sistemas de Inteligencia Artificial

Trabajo Práctico N°3

Perceptrón Simple y Multicapa



Alejo Flores Lucey	62622
Andrés Carro Wetzel	61655
Ian Franco Tognetti	61215
Matías Daniel Della Torre	61016

01 Introducción

Presentación del problema a resolver. Implementación

02 Perceptrón Simple Escalón

Resolución del problema de la función AND y XOR

03 Perceptrón Simple Lineal y No Lineal

Resolución de regresión con funciones LINEAR, LOGISTIC y TANH

04 Perceptrón Multicapa

04.01 Identificación de un dígito

Identificación de un número dibujado con 0s y 1s

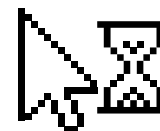
04.02 Paridad de un número

Resolución de sí un número recibido es par o impar

Introducción

Hemos implementado un **motor de redes neuronales** para resolver problemas utilizando el método de **aprendizaje supervisado**.

Hay problemas “simples” que pueden ser resueltos con una única neurona y otros problemas que son más complejos.



Perceptrón Simple Escalón

Es útil para problemas linealmente separables, donde se requiere una decisión de "sí o no".

Perceptrón Lineal y No Lineal

El perceptrón no lineal aprende patrones más complejos y resolver problemas no linealmente separables.

Perceptrón Multicapa

Red neuronal que consta de una capa de entrada, N capas ocultas y una capa de salida. Puede aprender relaciones complejas.

Métodos implementados

Activación

- Escalón
- Lineal
- Logística
- Tangente Hiperbólica

Métricas

- Accuracy
- Precision
- Recall
- F1-Score
- Mean Squared Error

Dataset Splitting

- K-Fold Cross Validation
- Shuffle Split
- Arbitrario

Optimización

- Gradiente Descendente
- Momentum
- Adam

Archivo de configuración

- Toda la configuración se maneja desde un archivo JSON.
- Se configura una semilla para obtener reproducibilidad en el caso de usar valores pseudo-aleatorios.
- Se elige el problema a resolver de las opciones disponibles.
- La arquitectura es completamente parametrizable. Se recibe un array de capas, donde se configura la función de activación y la cantidad de neuronas.

```
config.json
{
  "epsilon": 1e-5,
  "maxEpochs": 1000,
  "seed": 732,
  "learning": {
    "optimizer": {
      "type": "GRADIENT_DESCENT | MOMENTUM | ADAM",
      "options": {
        "rate": 0.1,
        "alpha": 0.8, // Only needed for MOMENTUM
        "beta1": 0.9, // Only needed for ADAM
        "beta2": 0.999 // Only needed for ADAM
      }
    },
    "updater": {
      "type": "BATCH | MINI-BATCH | ONLINE",
      "options": {
        "updateEveryXInputs": 15 // Only needed for MINI-BATCH
      }
    }
  },
  "perceptron": {
    "problem": "AND | XOR | SET | MULTILAYER_XOR | PARITY | DIGITS",
    "architecture": [
      {
        "neuronQty": 1,
        "activationFunction": {
          "type": "STEP | LINEAR | LOGISTIC | TANH",
          "options": {
            "beta": 1 // Only needed for LOGISTIC or TANH
          }
        }
      }
    ]
  }
}
```

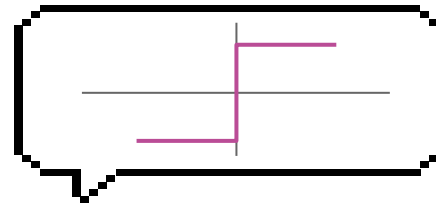
Detalles de implementación

Se hace uso del patrón **Singleton** para servir las constantes definidas en el archivo de configuración.

Se hace uso de **Numpy** para realizar operaciones matriciales y vectoriales con facilidad.

Se hace uso de **SKLearn** para la generación de métricas y para llevar adelante el *dataset splitting*.

Los **pesos de cada neurona** se inicializan en un **número aleatorio** entre -1 y 1, siguiendo una distribución uniforme.

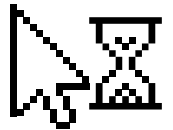


02

Perceptrón Simple Escalón

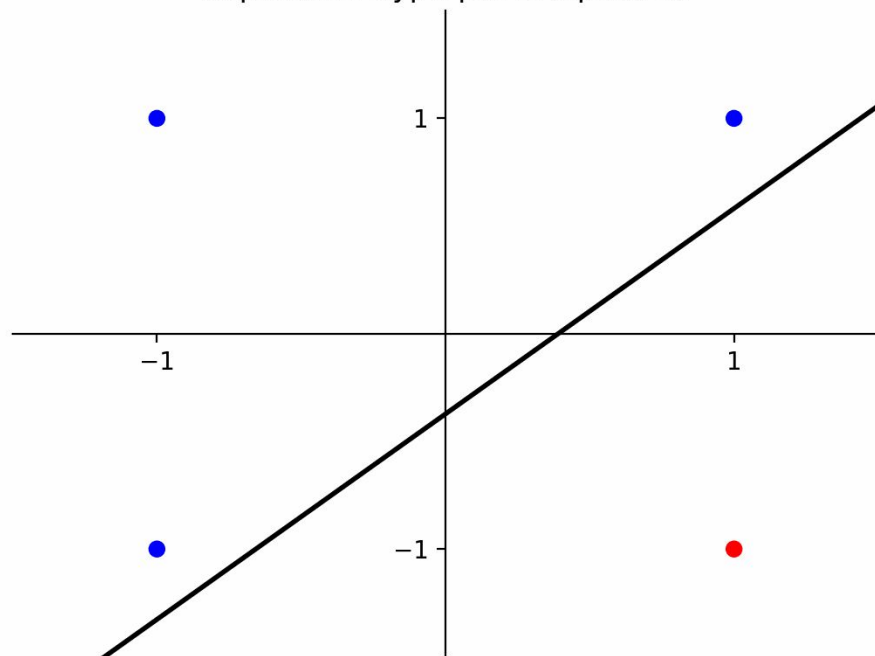
Con la implementación del perceptrón simple
analizar la resolución de algunos problemas lógicos
como **AND** y **XOR**

Función AND



Ajuste de pesos sinápticos

Separation Hyperplane (epoch 0)

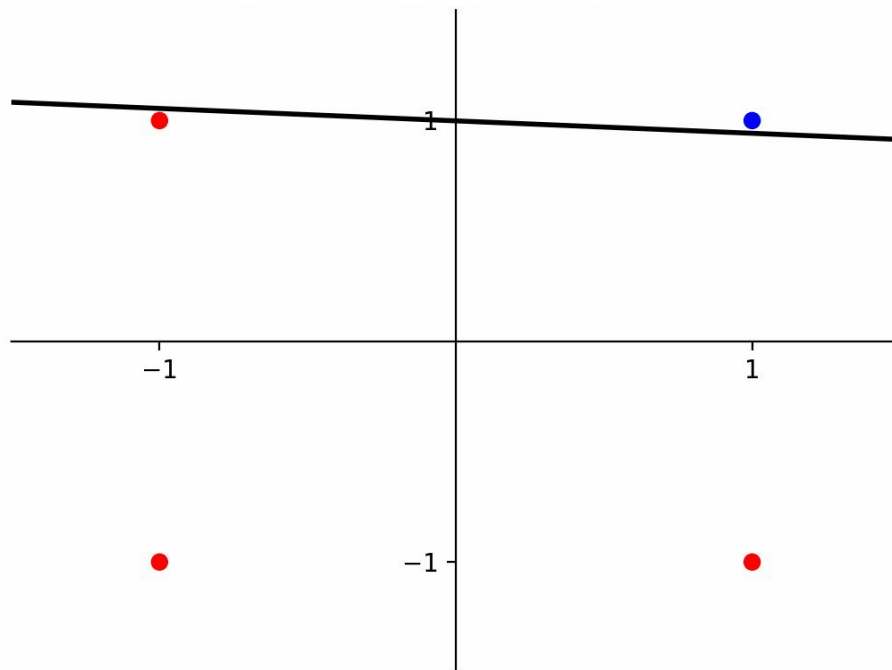


• True
• False

AND GATE

- Problem: **AND**
- $\epsilon =$ **0.1**
- Max Epochs: **10 000**
- Act.F.: **STEP**
- Optimizer:
GRADIENT DESCENT($\eta=1e-3$)
- Updater: **ONLINE**

Métricas



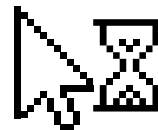
AND GATE

		Prediction	
		1	-1
Actual	1	1	0
	-1	0	3

- Accuracy: **1.0**
- Precision: **1.0**
- Recall: **1.0**
- F1-Score: **1.0**

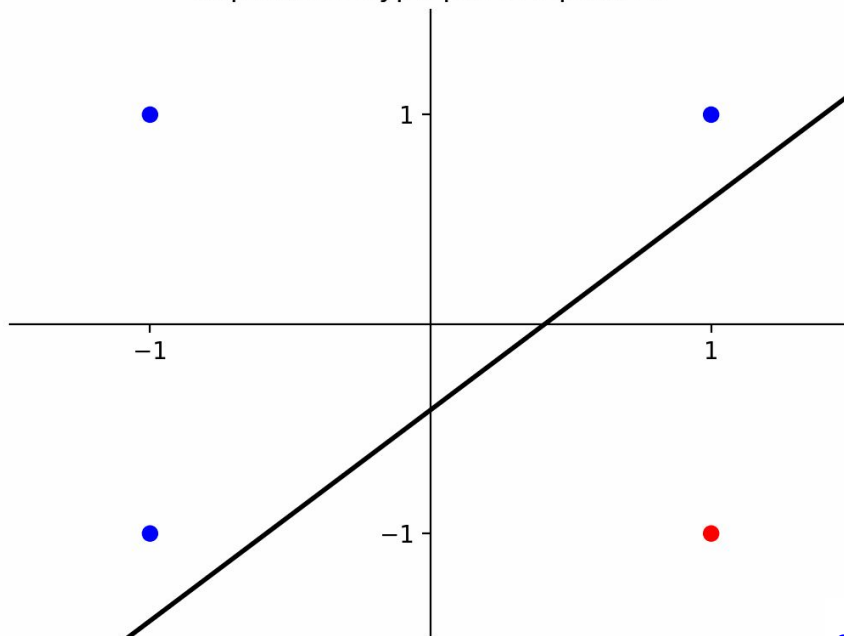
Función XOR

¿Se puede resolver con Perceptron Lineal Simple?



Ajuste de pesos sinápticos

Separation Hyperplane (epoch 1)



XOR GATE

• True
• False

- La compuerta XOR es un problema no linealmente separable.
- No se puede trazar una recta que separe los puntos en conjuntos esperados.
- No se puede resolver con un perceptrón lineal simple

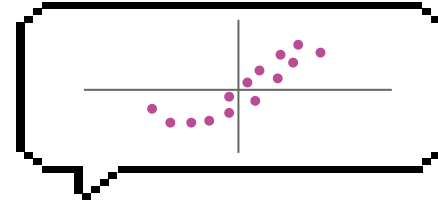
- Problem: **XOR**
- $\epsilon = 0.1$
- Max Epochs: **10 000**
- Act.F.: **STEP**
- Optimizer: **GRADIENT($\eta=1e-3$)**
- Updater: **ONLINE**

Métricas

No podemos determinar una recta de separación correcta para este caso pues no es posible resolverlo.

		Prediction	
		1	-1
Actual	1	0	1
	-1	2	1

- Accuracy: 0.25
- Precision: 0.166
- Recall: 0.25
- F1-Score: 0.2



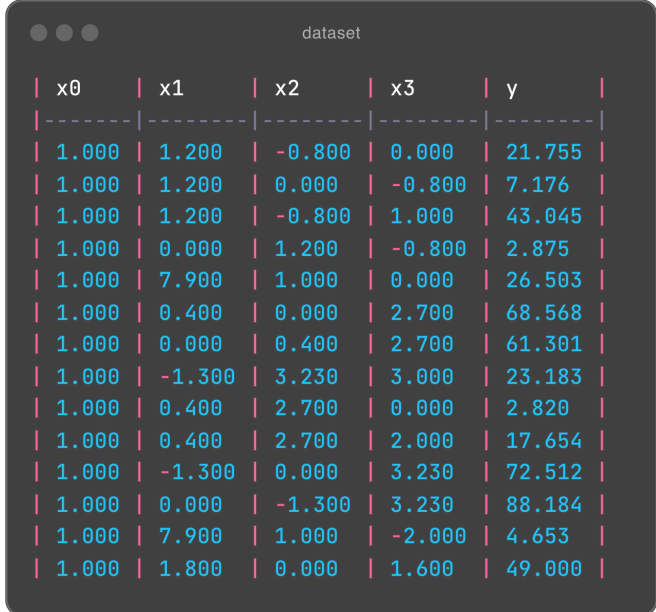
03

Perceptrón Simple Lineal y No Lineal

Sirve para resolver problemas de regresiones.
Estudiaremos el comportamiento de las diferentes
funciones de activación para un dataset desconocido.

Problema

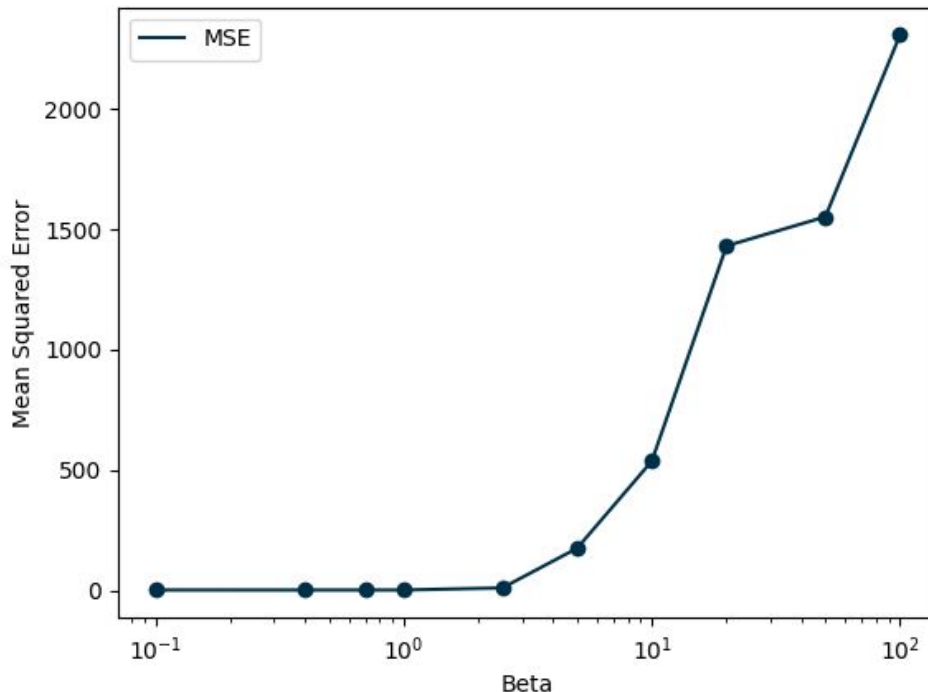
- ¡Es un problema de regresión!
- El objetivo es encontrar si se ajusta a una regresión lineal o sigmoidal.
- ¿Afectará la manera de partir el dataset en training y testing a la hora de resolver el problema?



x0	x1	x2	x3	y
1.000	1.200	-0.800	0.000	21.755
1.000	1.200	0.000	-0.800	7.176
1.000	1.200	-0.800	1.000	43.045
1.000	0.000	1.200	-0.800	2.875
1.000	7.900	1.000	0.000	26.503
1.000	0.400	0.000	2.700	68.568
1.000	0.000	0.400	2.700	61.301
1.000	-1.300	3.230	3.000	23.183
1.000	0.400	2.700	0.000	2.820
1.000	0.400	2.700	2.000	17.654
1.000	-1.300	0.000	3.230	72.512
1.000	0.000	-1.300	3.230	88.184
1.000	7.900	1.000	-2.000	4.653
1.000	1.800	0.000	1.600	49.000

Ajuste No Lineal

Suponemos que los datos se ajustan con la función LOGISTIC.
¿Qué valor de Beta es el adecuado?



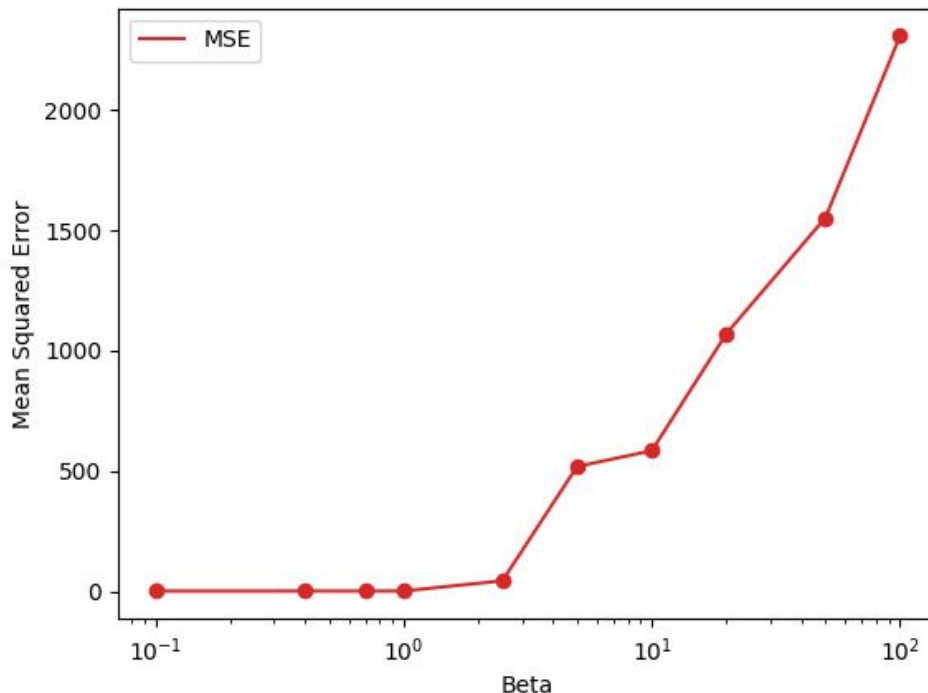
Conclusión: $\beta = 0.4$

- Problem: SET
- $\epsilon = 3e-4$
- Max Epochs: 20 000
- Act.F.: LOGISTIC($\beta=?$)
- Optimizer: GRADIENT DESCENT($\eta=5e-4$)
- Updater: ONLINE

Ajuste No Lineal

Suponemos que los datos se ajustan con la función TANH .

¿Qué valor de β es el adecuado?



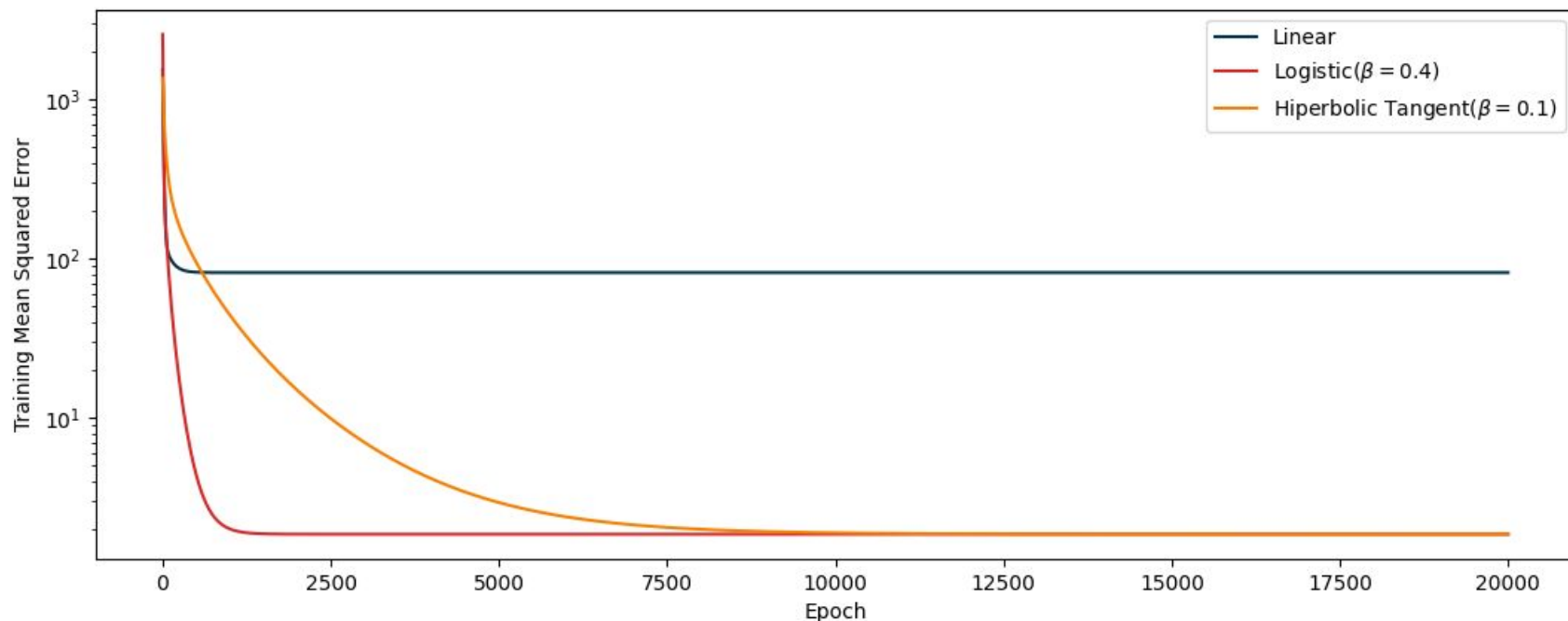
Conclusión: $\beta = 0.1$

- Problem: SET
- $\epsilon = 3e-4$
- Max Epochs: 20 000
- Act.F.: $\text{TANH}(\beta=?)$
- Optimizer: GRADIENT DESCENT($\eta=5e-4$)
- Updater: ONLINE

Poder de Aprendizaje

Habiendo obtenido el mejor Beta para LOGISTIC y TANH, queremos ver **qué función de regresión ajusta mejor al dataset.**

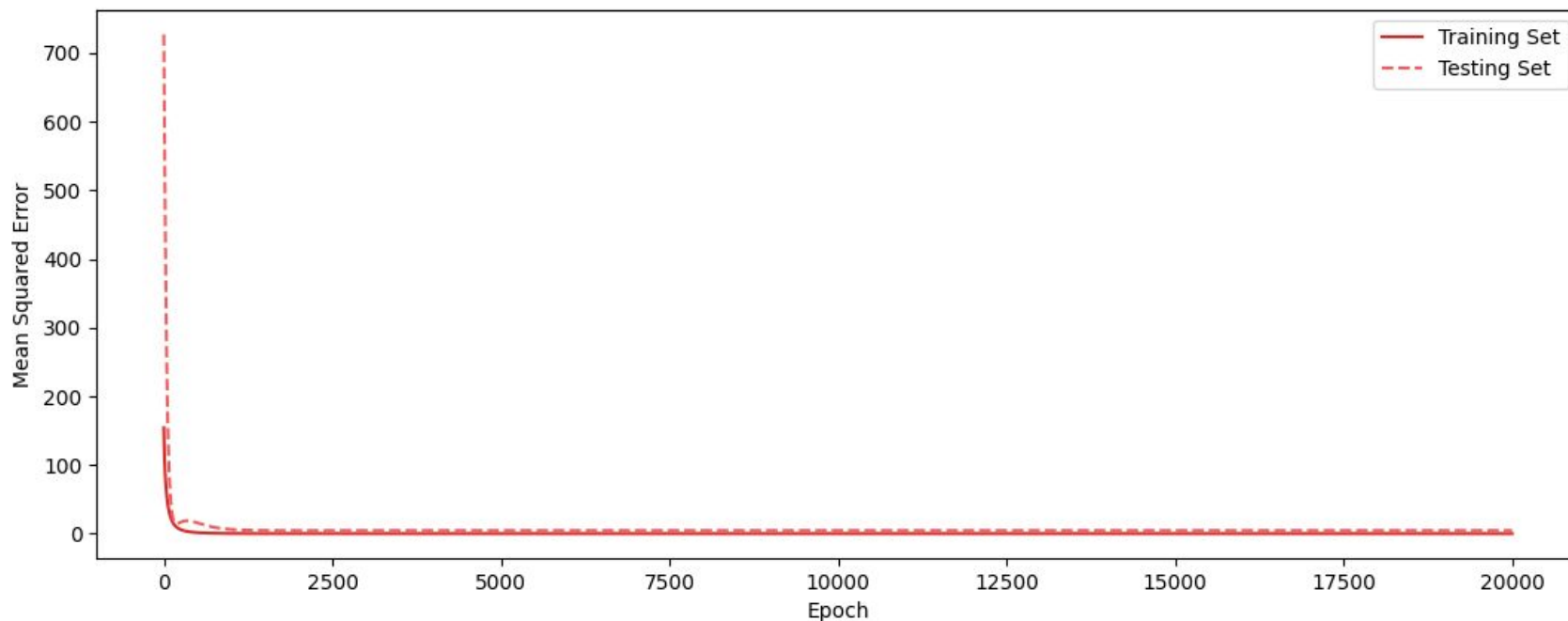
- Problem: **SET**
- $\epsilon = 3e-4$
- Max Epochs: **20 000**
- Act.F.: **...**
- Optimizer: **GRADIENT DESCENT($\eta=5e-4$)**
- Updater: **ONLINE**



Poder de Generalización

Ya sabemos que los datos se ajustan con la función LOGISTIC.
Ahora, ¿es posible generalizar?

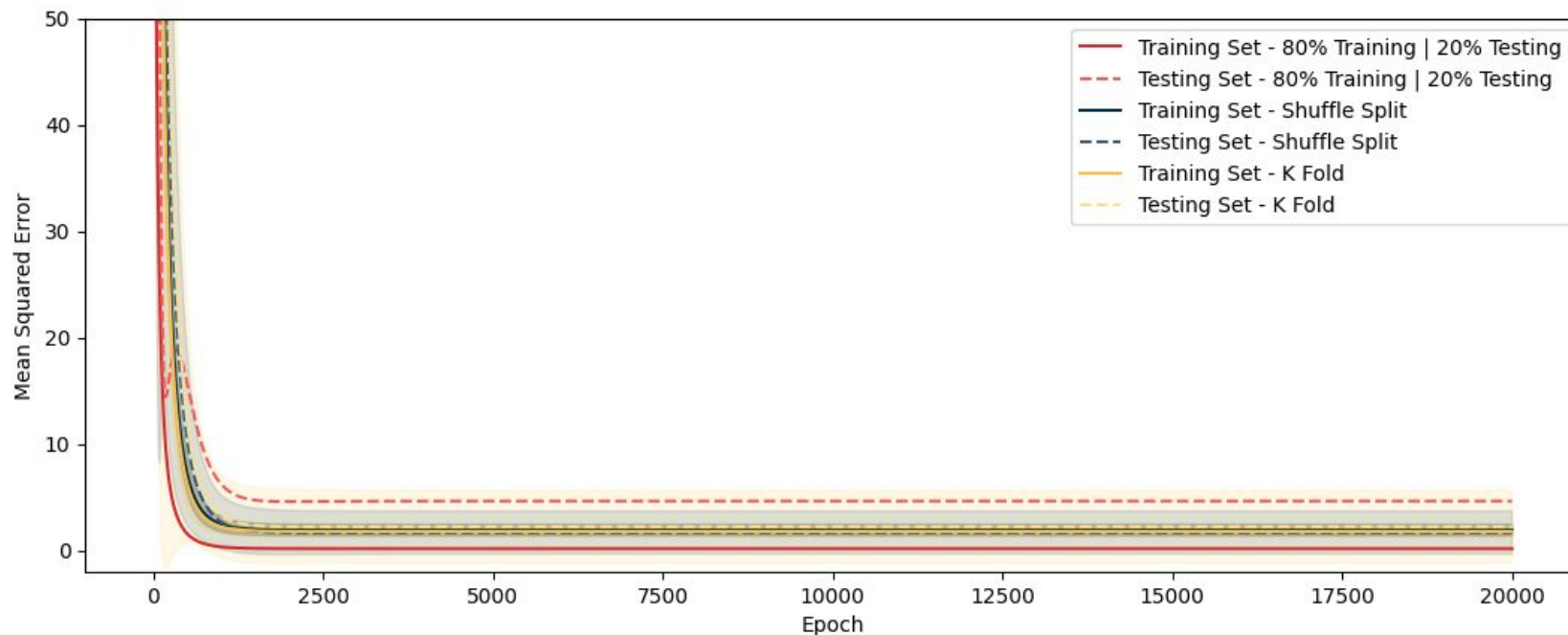
- Problem: SET
- $\epsilon = 3e-4$
- Max Epochs: 20 000
- Act.F.: LOGISTIC($\beta=0.4$)
- Optimizer: GRADIENT DESCENT($\eta=5e-4$)
- Updater: ONLINE



Poder de Generalización

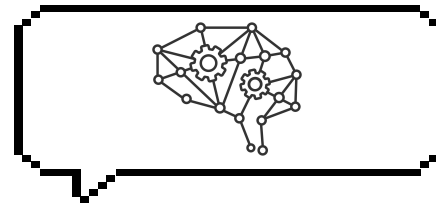
¿Afecta el **modo de separar el conjunto** de training y testing para el poder de generalización?

- Problem: **SET**
- $\epsilon = 3e-4$
- Max Epochs: **20 000**
- Act.F.: **LOGISTIC($\beta=0.4$)**
- Optimizer:
GRADIENT DESCENT($\eta=5e-4$)
- Updater: **ONLINE**



Conclusiones

- El hiperparámetro β de la función LOGISTIC y TANH debe pertenecer al rango $(0,1]$.
- El dataset proporcionado presenta datos no lineales.
- La regresión se ajusta más rápido con la función de activación LOGISTIC.
- El perceptrón no lineal tiene la capacidad de generalizar.
- El mejor método para partir el dataset en conjunto de training y testing es Shuffle Split.

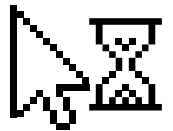


04

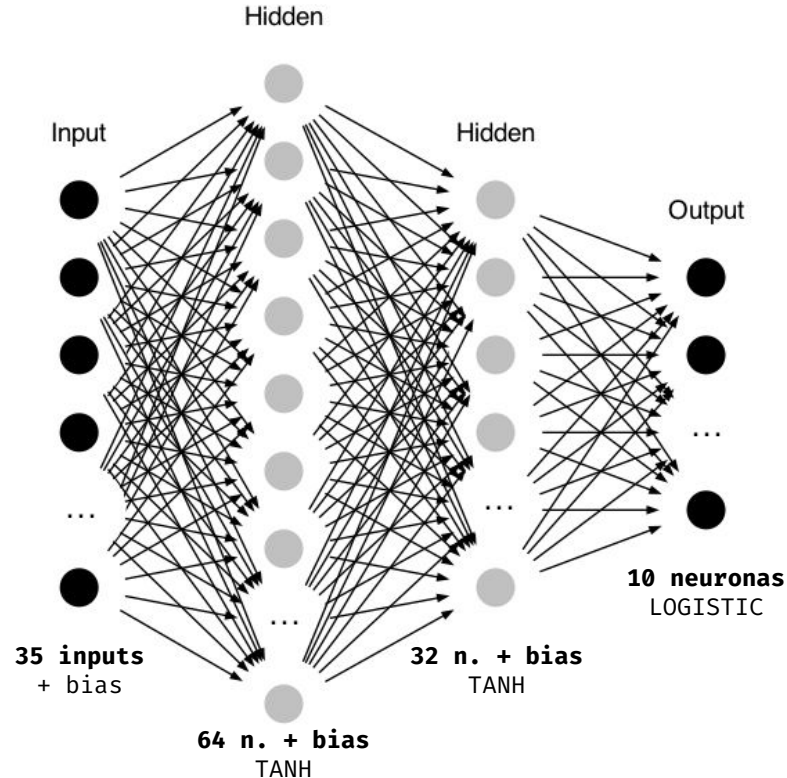
Perceptrón Multicapa

Para resolver problemas más complejos como la identificación de dígitos y la paridad, se utilizó un perceptrón multicapa

Identificador de dígitos



Arquitectura

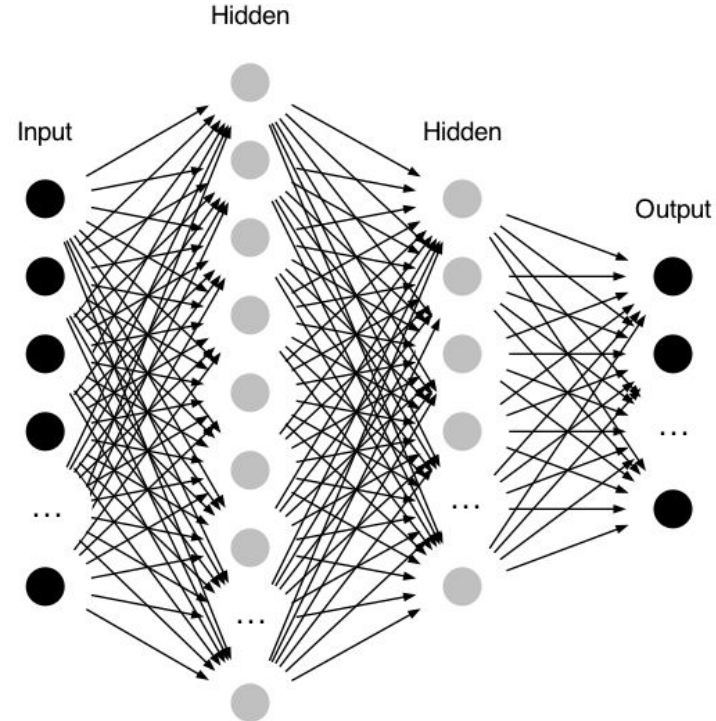


Training

Se entrena la red con un dataset de los dígitos del 0 al 9.

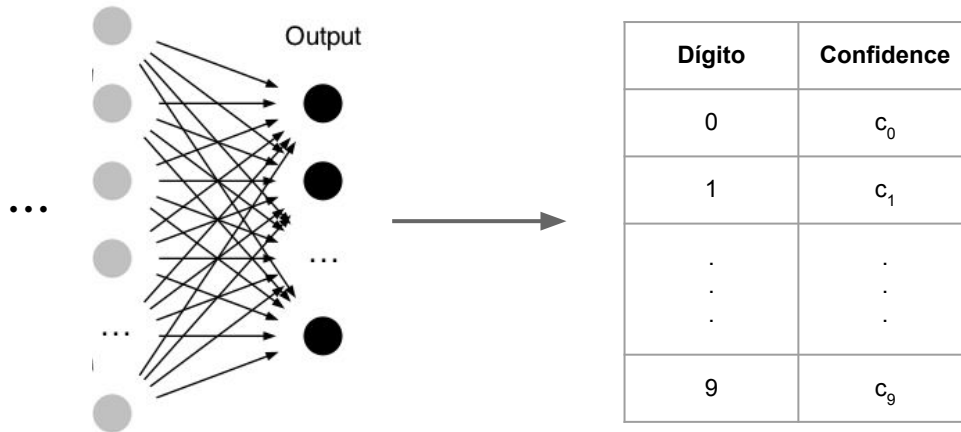
	1	2	3	4	5
1	0	0	1	0	0
2	0	1	1	0	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	0	0
6	0	0	1	0	0
7	0	1	1	1	0

Representación
sin ruido



Testing

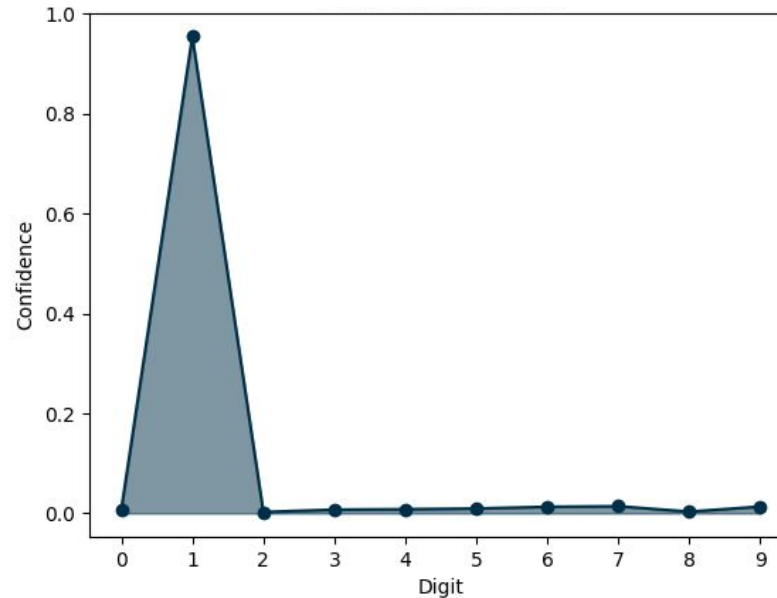
- Input: dataset **desconocido** de dígitos del 0-9 **con ruido gaussiano**.
- Cada neurona de salida identifica a un dígito del 0 al 9.
- La red devuelve una lista con la seguridad de que el número de input corresponda a cada uno de los dígitos 0-9.



El índice de la fila con mayor seguridad será el dígito que está prediciendo la red neuronal.

¿Cómo funciona el perceptrón?

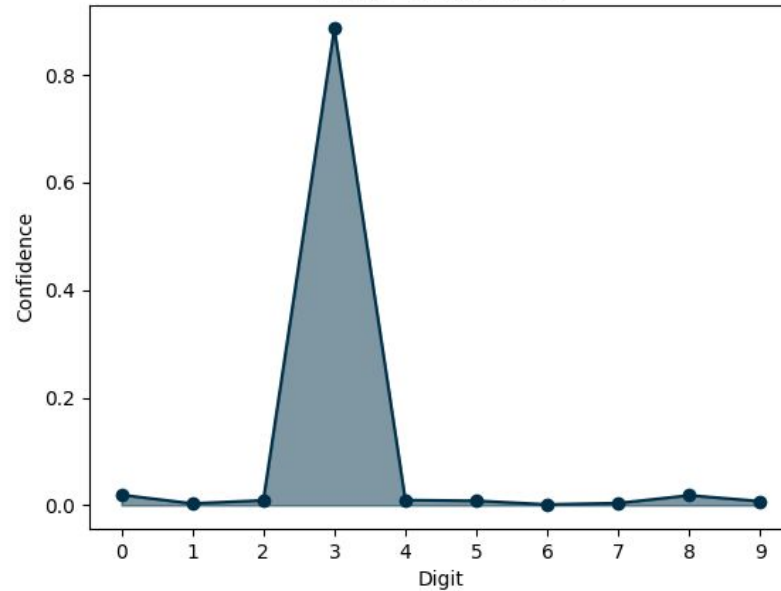
	1	2	3	4	5
1	0	0	1	0	0
2	0	1	1	0	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	0	0
6	0	0	1	0	0
7	0	1	1	1	0



1

¿Cómo funciona el perceptrón?

	1	2	3	4	5
1	0	1	1	1	0
2	1	0	0	0	1
3	0	0	0	0	1
4	0	0	1	1	0
5	0	0	0	0	1
6	1	0	0	0	1
7	0	1	1	1	0

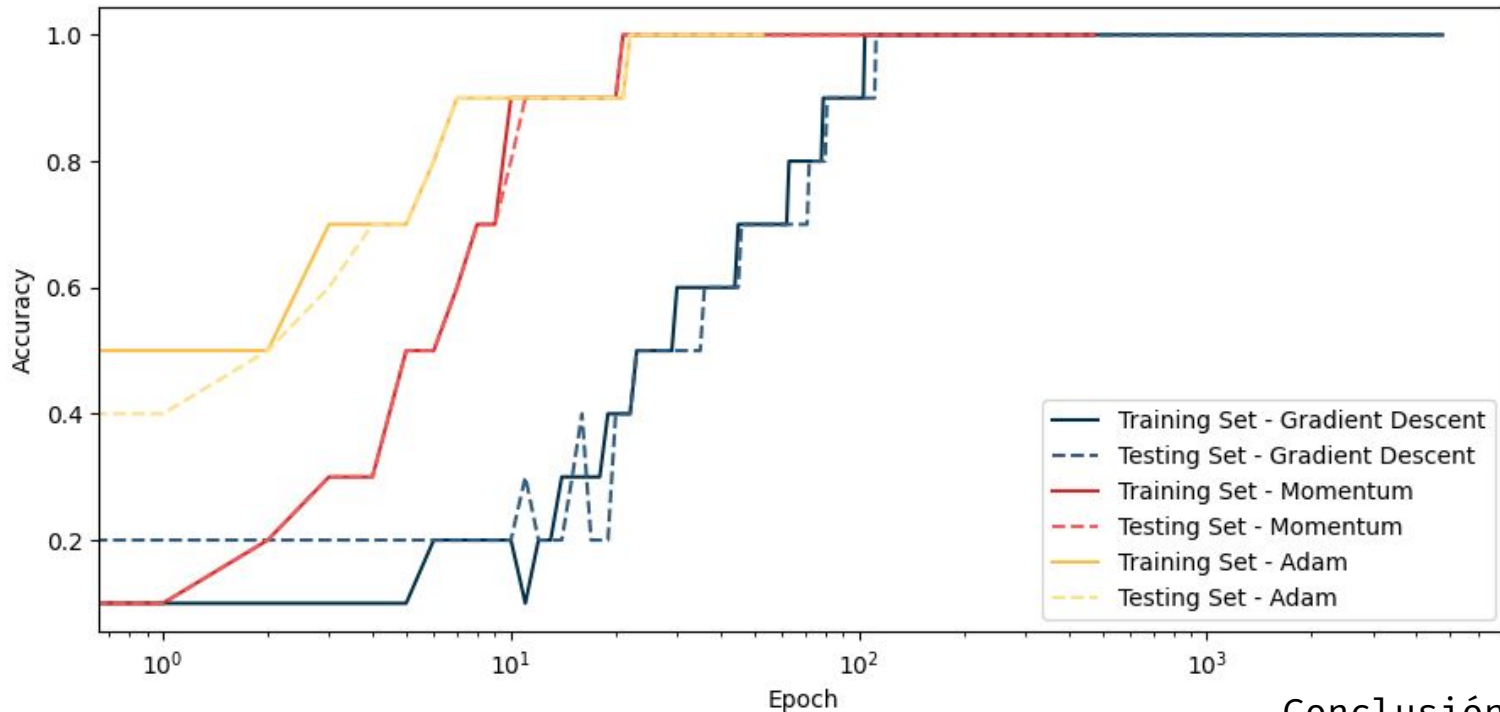


3

Elección de hiperparámetros

¿Qué método de optimización resuelve mejor el problema?

- Problem: **DIGITS**
- $\epsilon = 3e-4$
- Max Epochs: **10 000**
- Optimizer: **?**
- Updater: **ONLINE**



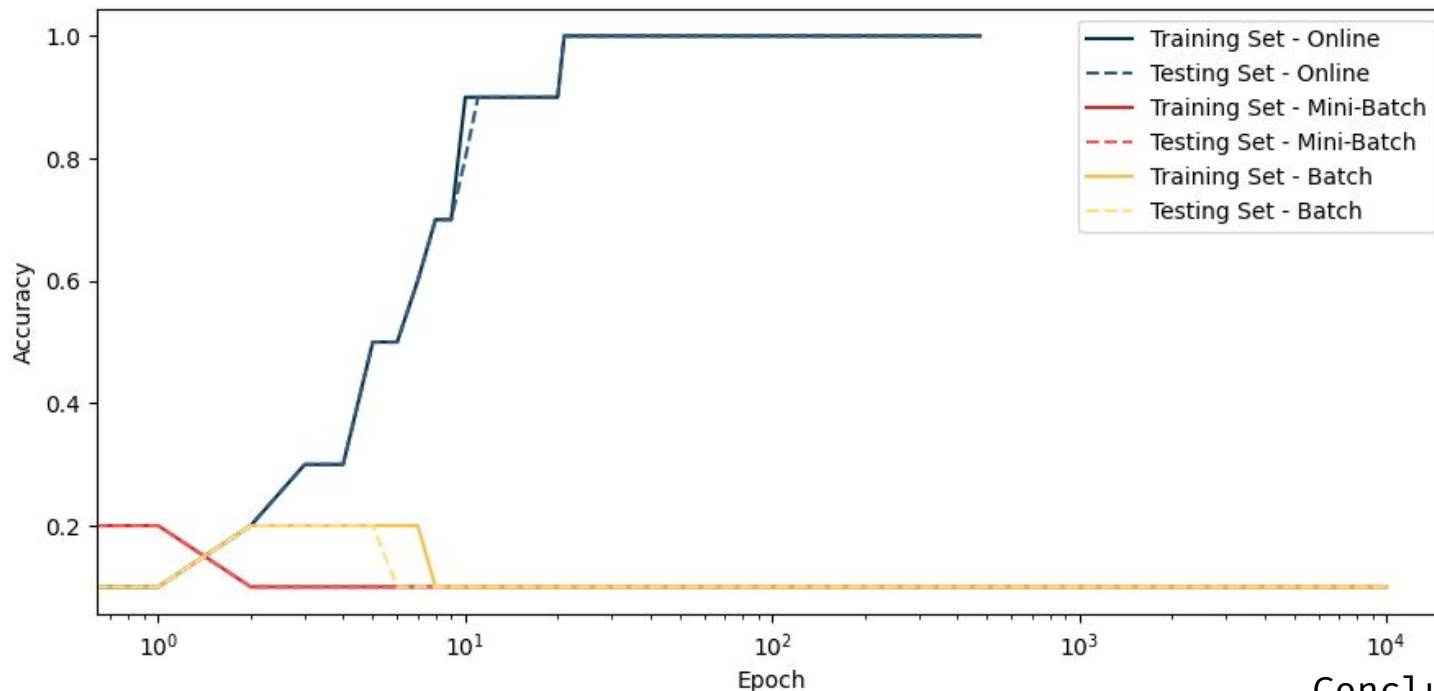
$\eta = 1e-2$
 $\alpha = 0.9$
 $\beta_1 = 0.9$
 $\beta_2 = 0.999$

Conclusión: **Momentum**

Elección de hiperparámetros

¿Qué **método de actualización** resuelve mejor el problema?

- Problem: **DIGITS**
- $\epsilon = 3e-4$
- Max Epochs: **10 000**
- Optimizer: **MOMENTUM($\eta=1e-2, \alpha=0.9$)**
- Updater: **?**

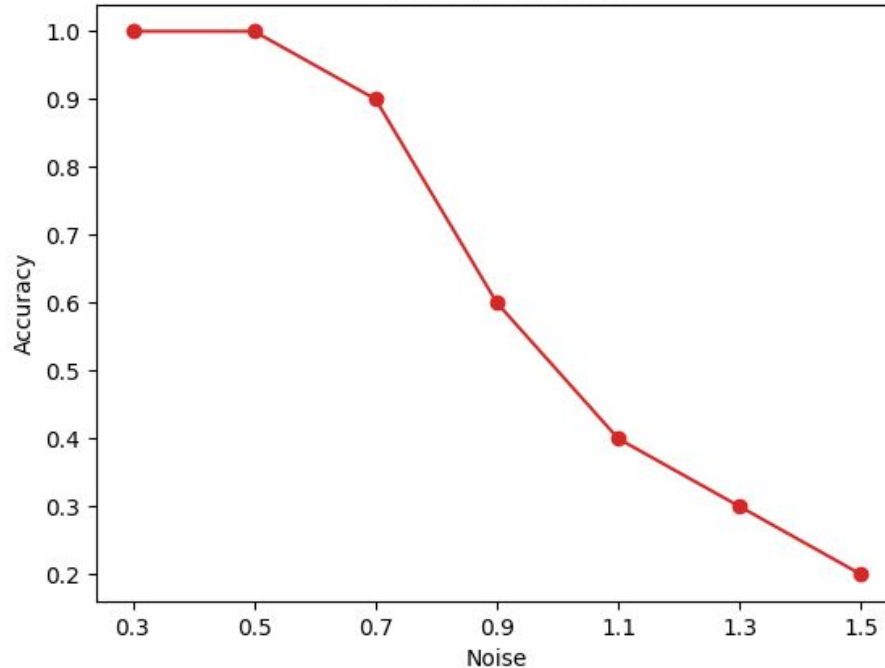


Conclusión: **Online**

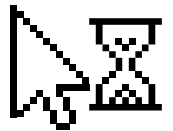
¿Cómo afecta el ruido?

Si testeamos el perceptrón con dígitos con más o menos ruido gaussiano, ¿que ocurre?

- Problem: **DIGITS**
- $\epsilon = 3e-4$
- Max Epochs: **10 000**
- Optimizer: **MOMENTUM($\eta=1e-2, \alpha=0.9$)**
- Updater: **ONLINE**



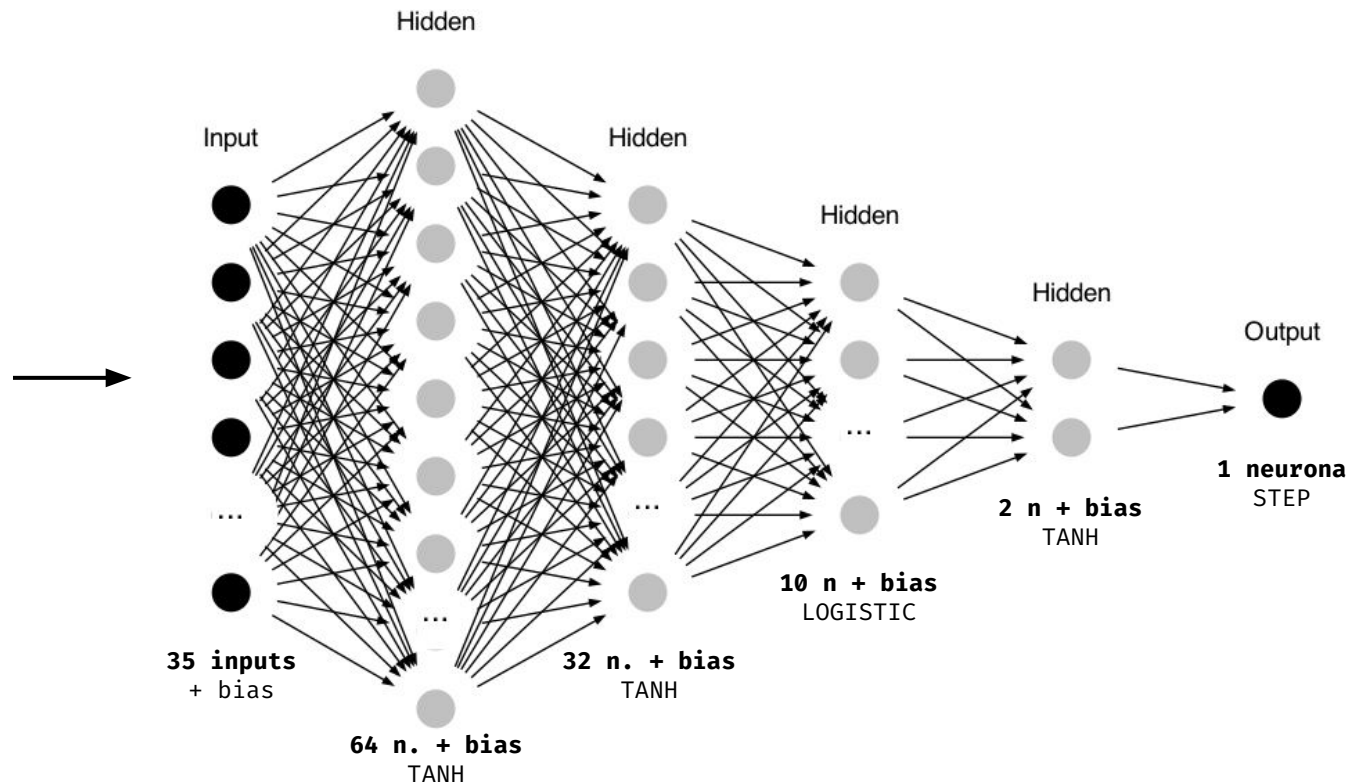
Paridad de un número



Arquitectura

	1	2	3	4	5
1	0	0	1	0	0
2	0	1	1	0	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	0	0
6	0	0	1	0	0
7	0	1	1	1	0

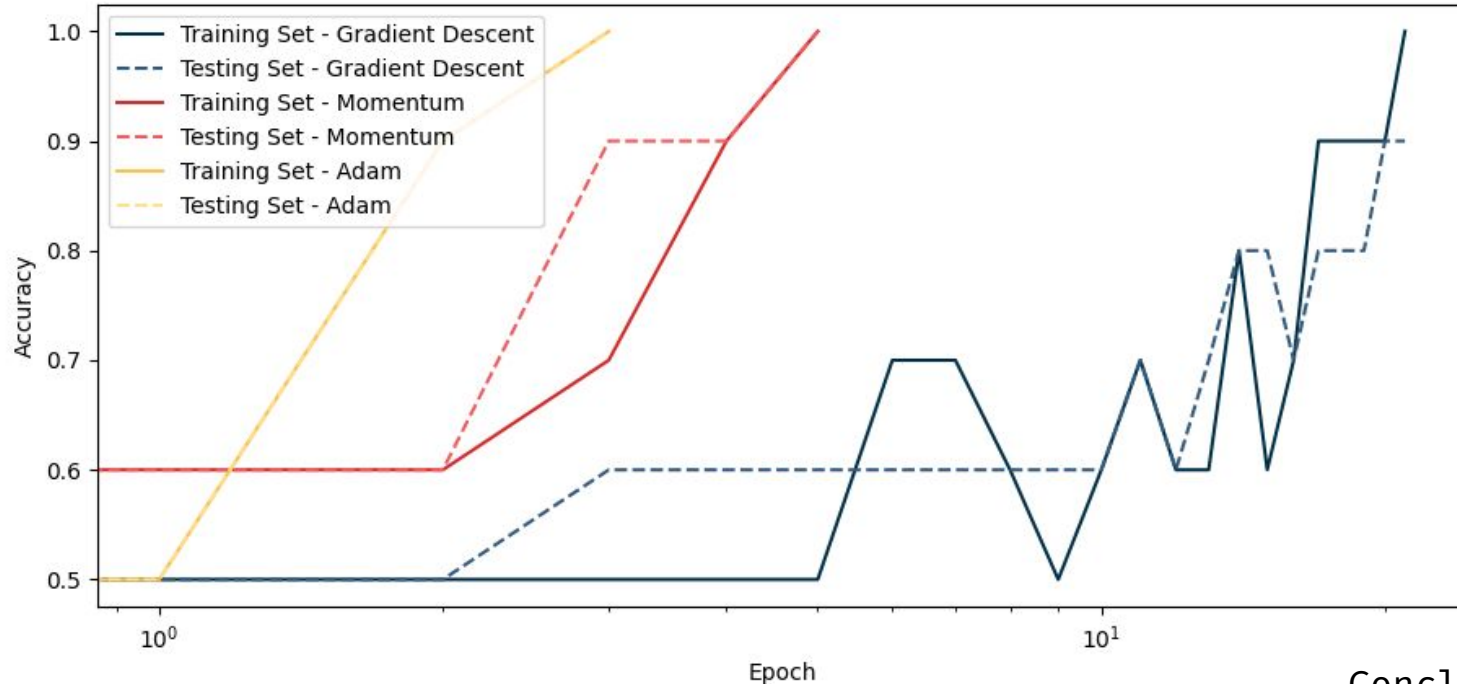
Representación
del dígito



Elección de hiperparámetros

¿Qué **método de optimización** resuelve mejor el problema?

- Problem: **PARITY**
- $\epsilon = 3e-4$
- Max Epochs: **10 000**
- Optimizer: **?**
- Updater: **ONLINE**



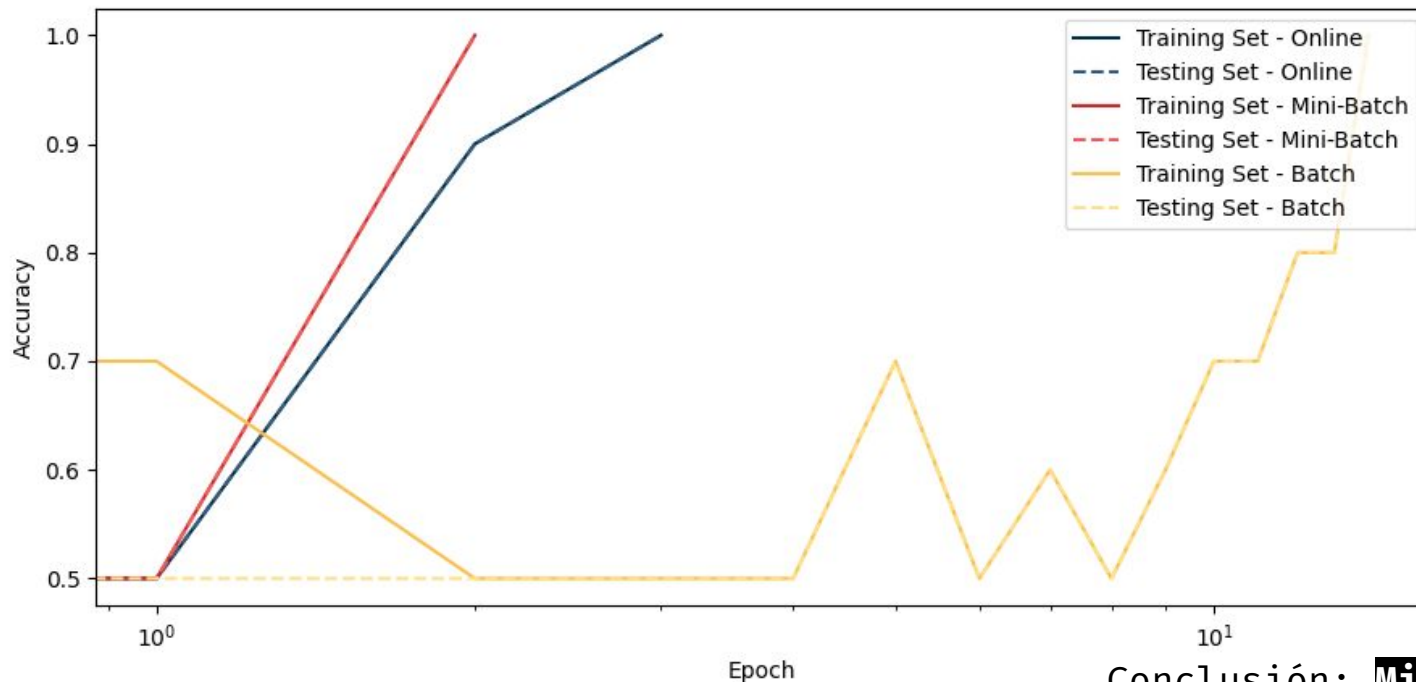
$\eta = 1e-2$
 $\alpha = 0.9$
 $\beta_1 = 0.9$
 $\beta_2 = 0.999$

Conclusión: **Adam**

Elección de hiperparámetros

¿Qué método de actualización resuelve mejor el problema?

- Problem: **PARITY**
- $\epsilon = 3e-4$
- Max Epochs: **10 000**
- Optimizer: **ADAM($\eta=1e-2$, $\beta_1=0.9$, $\beta_2=0.999$)**
- Updater: **?**

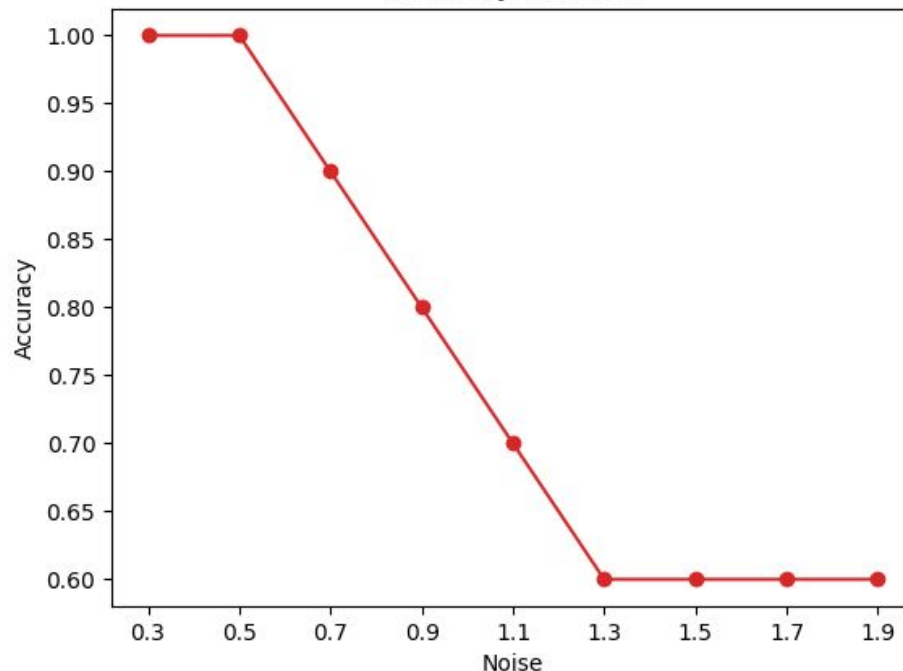


Conclusión: **Mini Batch(x=2)**

¿Cómo afecta el ruido?

Si testeamos el perceptrón con dígitos con más o menos ruido gaussiano, ¿que ocurre?

- Problem: **PARITY**
- $\epsilon = 3e-4$
- Max Epochs: **10 000**
- Optimizer: **ADAM($\eta=1e-2$, $\beta_1=0.9$, $\beta_2=0.999$)**
- Updater: **MINI-BATCH(x=2)**



Conclusiones

- El método Adam llega, por lo general, más rápido a ser exacto. Momentum lo sigue.
- Gradiente Descendente tarda varias épocas más que los otros métodos.
- Con respecto a los métodos de actualización, el método Online siempre logra ser exacto dentro de los estudios que realizados.
- A medida que se incrementa el ruido en el training set, más imprecisa es la accuracy del perceptrón

¡Muchas gracias por su atención!

CREDITS: This presentation template was created by
Slidesgo, and includes icons by **Flaticon**, and infographics
& images by **Freepik**

Enlaces Útiles

- <https://github.com/alejofl/sia>
Repositorio del proyecto.
- <https://github.com/martisak/dotnets>
Generador de imágenes de una red neuronal feed-forward.
- <https://mxnet.apache.org/versions/1.5.0/tutorials/python/mnist.html>
Implementación de reconocimiento de dígitos.
- <https://medium.com/analytics-vidhya/xor-gate-with-multilayer-perceptron-66e78671acd4>
Implementación de XOR en un perceptrón multicapa.