



Perceptrones

- Banfi, Malena
- Fleischer, Lucas
- Perez Rivera, Mateo
- Szejer, Ian

Ejercicio 2

Consigna

Implementar el algoritmo de perceptrón simple lineal y perceptrón simple no lineal. Utilizar ambos para aprender a clasificar los datos en el archivo "TP2-ej2-conjunto.csv".



¿Cómo medimos el error del perceptrón lineal y no lineal?

Hacemos la sumatoria del valor absoluto de las diferencias entre los resultados obtenidos y los esperados. Luego dividimos por el tamaño del conjunto de testeo.

Además, mostraremos los resultados para que se analicen visualmente.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$



»»» Perceptrón No Lineal: Capacidad de generalización



Capacidad de generalización

Es la capacidad de obtener buenos resultados con datos que no estén ni sean parte del conjunto que se va a usar para entrenar.



Factor de aprendizaje

Determina cuanto se modifican los pesos en cada paso.



División de datos de entrada

Se dividió los datos de entrada en 2 subconjuntos:

- Entrenamiento y prueba



Selección de resultados

Calculamos el error del conjunto de testeo y se selecciona el más chico.

Evaluación de la capacidad de generalización del perceptrón simple no lineal utilizando, de los datos provistos, un subconjunto para entrenar y otro para testear.

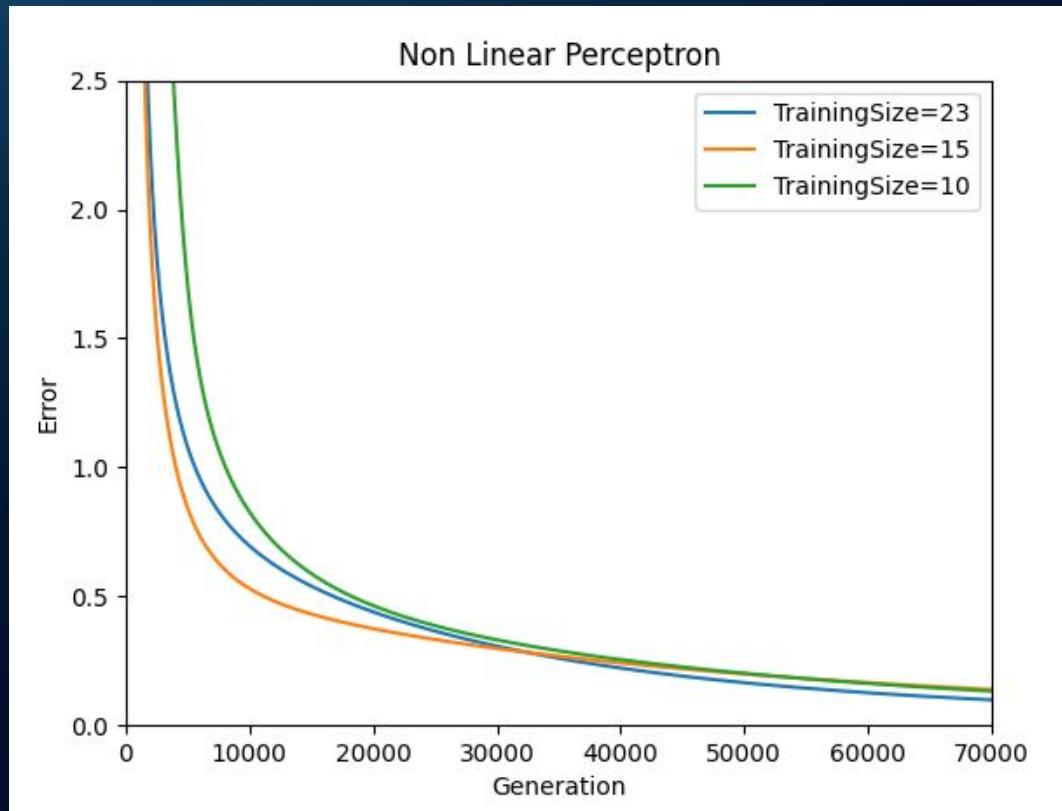
- Para conseguir el mejor conjunto de entrenamiento es necesario realizar una validación cruzada del modelo.
 - Dividir los datos en un conjunto de entrenamiento y un conjunto de validación, y luego entrenar el modelo en el conjunto de entrenamiento y evaluar su rendimiento en el conjunto de validación.
 - Este proceso se repite varias veces, cambiando los conjuntos de entrenamiento y validación en cada iteración.
-
- La validación cruzada permite estimar el rendimiento del modelo en datos no vistos y seleccionar los hiperparámetros del modelo que producen el mejor rendimiento en promedio en todos los conjuntos de entrenamiento y validación.
 - Se puede evitar el sobreajuste del modelo y obtener una buena generalización en datos nuevos.



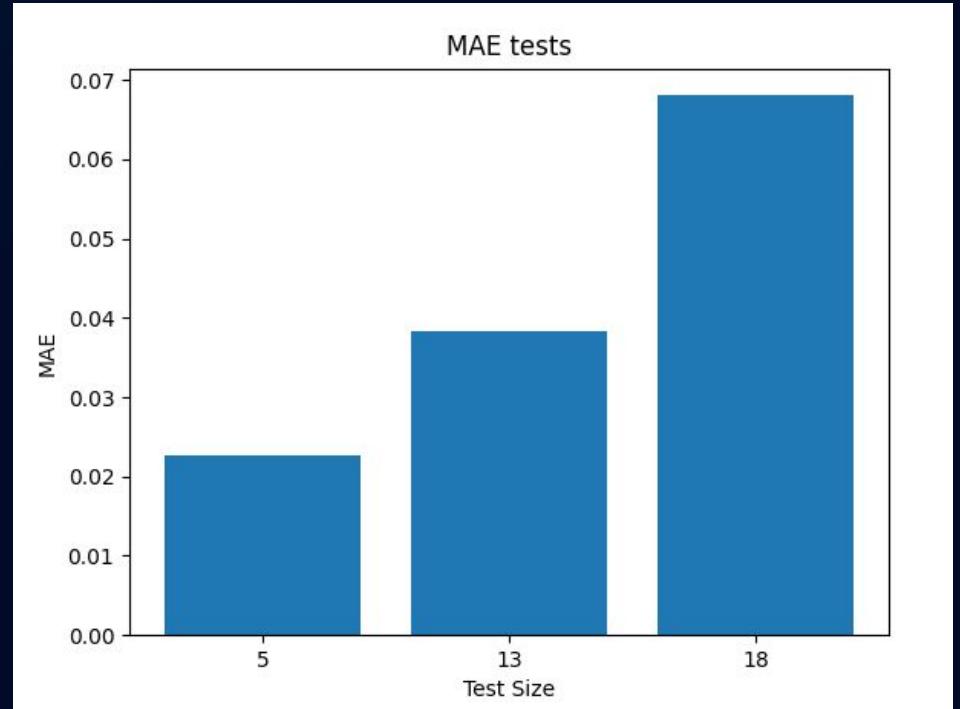
Capacidad de generalización

Resultados

Error en cuanto a generaciones variando el training size

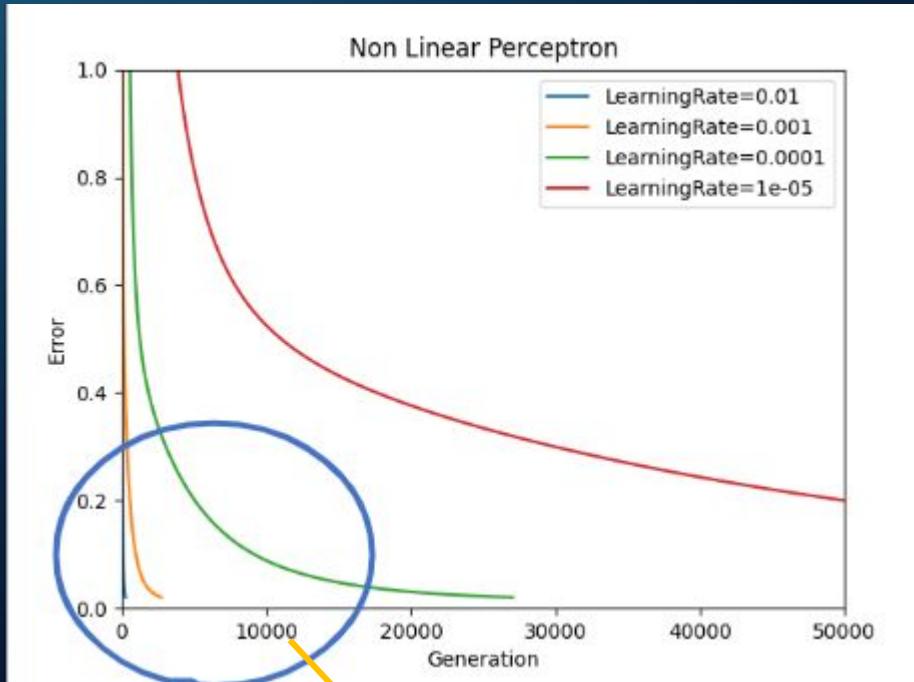


Learning rate: 0.0001
Beta: 0.8
Min_error: 0.02
Generations: 70000

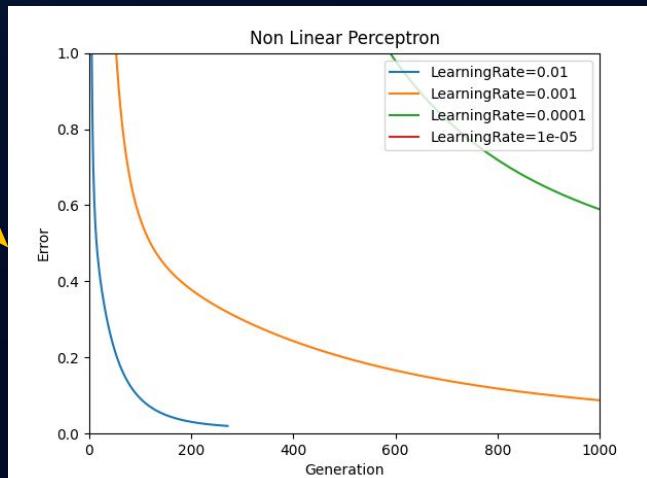


- Los mejores porcentajes para entrenar y testear son 70%/30% a 80%/20%.
- A mayor conjunto de entrenamiento, más aprende y por lo tanto tendrá menor error el cjto de testeo.

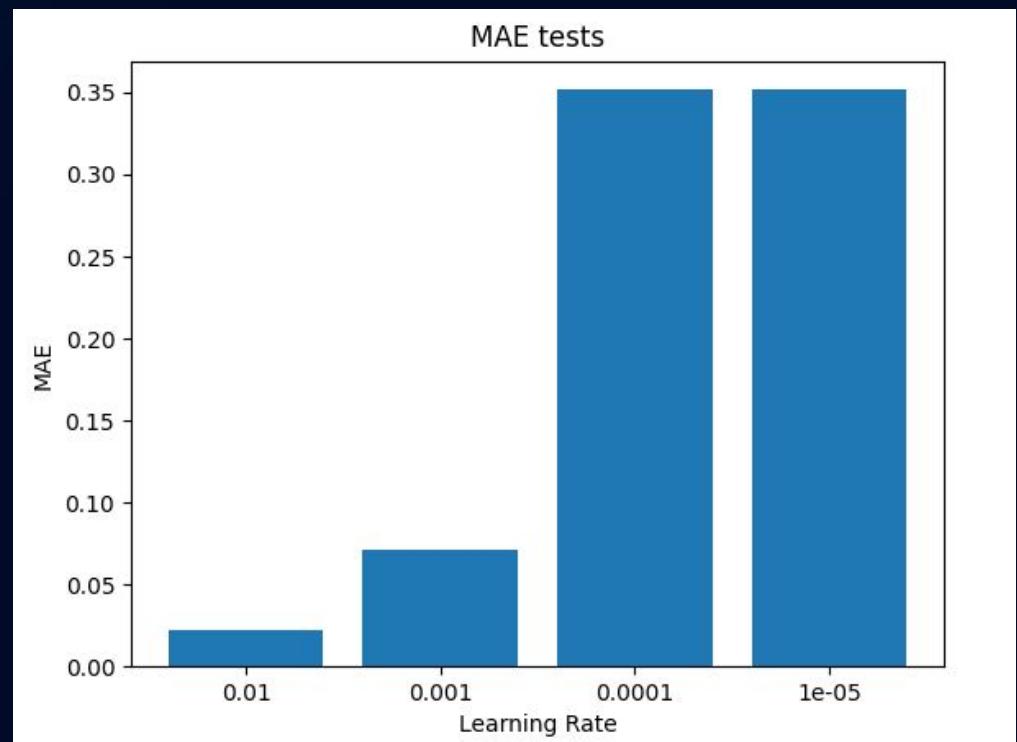
Error en cuanto a generaciones variando el learning rate



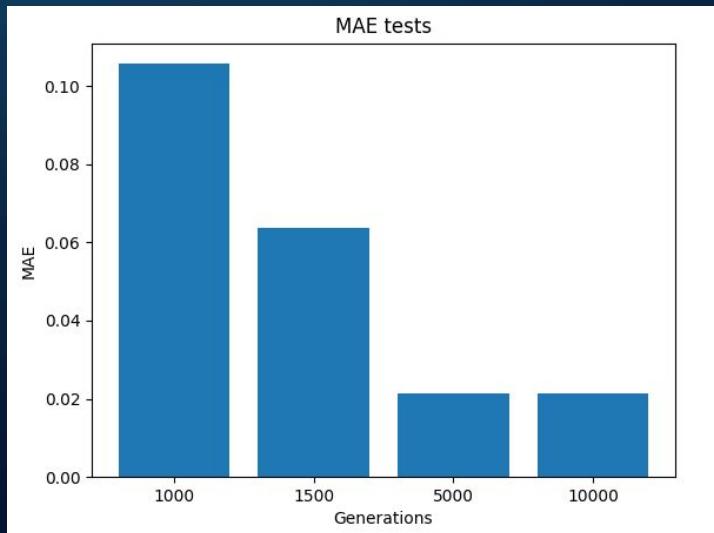
zoom



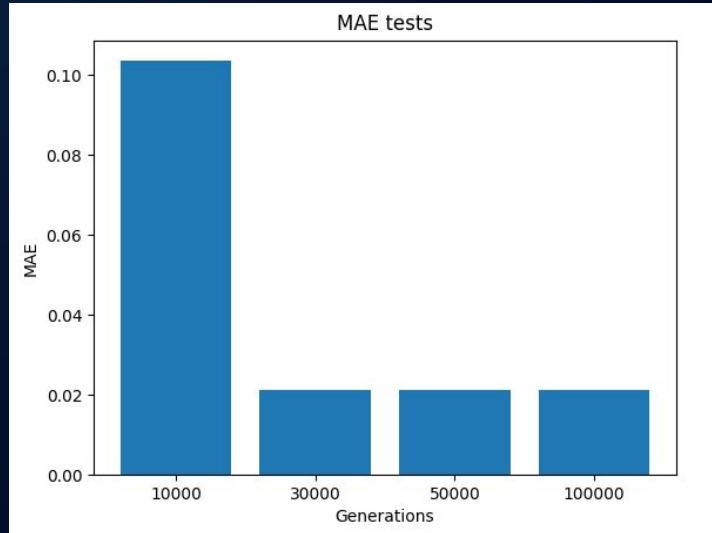
Training size: 23 (80%
entrenamiento)
Beta: 0.8
Min_error: 0.02
Generations: 50000



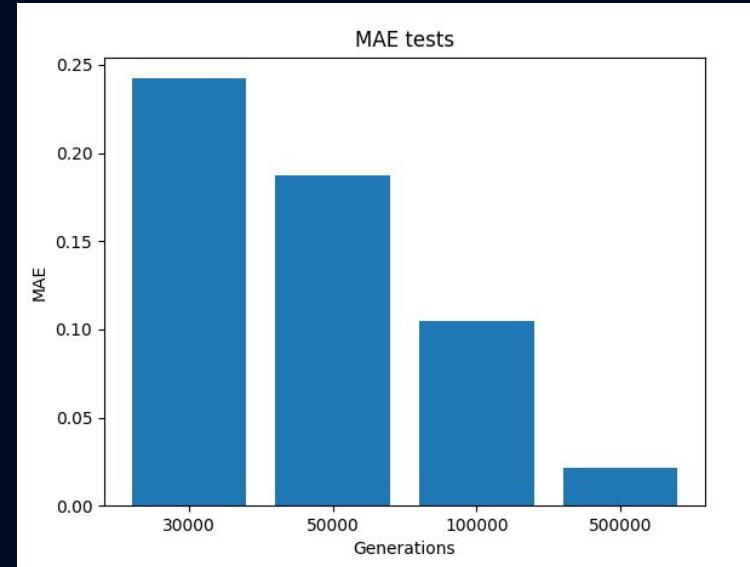
Error en cuanto a la cantidad de generaciones



Training size: 23
Beta: 0.8
Min_error: 0.02
Learning rate: 0.01



Training size: 23
Beta: 0.8
Min_error: 0.02
Learning rate: 0.001



Training size: 23
Beta: 0.8
Min_error: 0.02
Learning rate: 0.0001

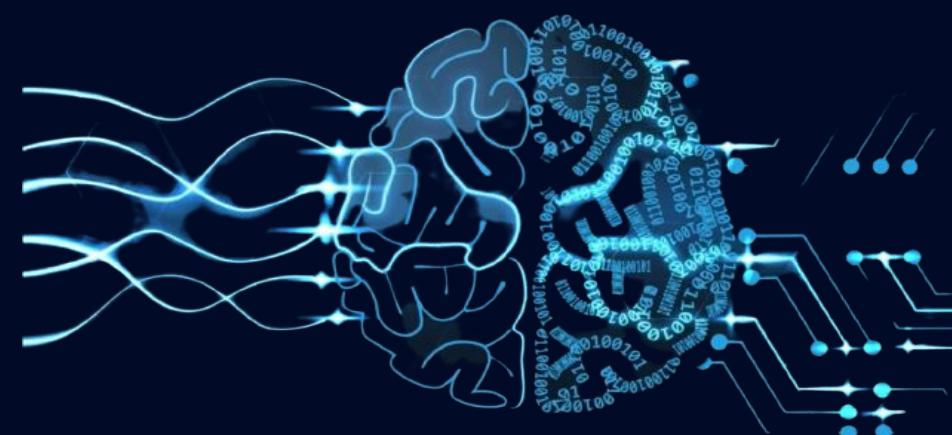
»» Conclusion: ¿Cómo elegirían el mejor conjunto de entrenamiento?



Learning rate: 0.01

**Tamaño del
entrenamiento/test
eo: 80/20%**

**Cantidad de
generaciones: 50.000**



»»» ¿Qué efecto tiene la elección en la capacidad de generalización del perceptrón?



Cuando generalizamos, queremos que el conjunto de entrenamiento aprenda para que al testear datos que jamás vio, de un mejor resultado.

Pudimos ver en los resultados anteriores, que con dichos hiperparametros, obtenemos los mejores (mas pequeños) errores. A menor error, mejores resultados uno ve.



¿Que es la cross k validation?

El objetivo de la validación cruzada es estimar cómo se comportaría un modelo en un conjunto de datos no visto, al mismo tiempo que se evita el sobreajuste en el conjunto de datos de entrenamiento.

Existen diferentes tipos de validación cruzada, como la validación cruzada k-fold, la validación cruzada estratificada k-fold, la validación cruzada leave-one-out y la validación cruzada de repetición aleatoria. El método k-fold es uno de los más utilizados, y consiste en dividir los datos en k pliegues, donde k es un número entero predeterminado. Luego, se entrena el modelo en $k-1$ pliegues y se evalúa en el pliegue restante. Este proceso se repite k veces, de tal manera que cada pliegue se utiliza para evaluar el modelo una vez.

Finalmente, se calcula la media de las métricas de rendimiento para obtener una estimación del rendimiento del modelo en datos no vistos.

Nos pareció que teníamos un conjunto de datos muy pequeño como para que esta validación sea relevante.



Conclusiones del perceptrón no lineal

El perceptrón no lineal tiene la capacidad de resolver problemas no lineales, lo que lo hace adecuado para una amplia gama de aplicaciones en las que los datos pueden no estar linealmente separados. Por esto es que el mismo puede aprender la función de entrada sin problema a diferencia del perceptrón lineal.

Ejercicio 3

Consigna

Implementar el algoritmo de perceptrón multicapa y utilizarlo para aprender.



Ejercicio 3 A

Consigna

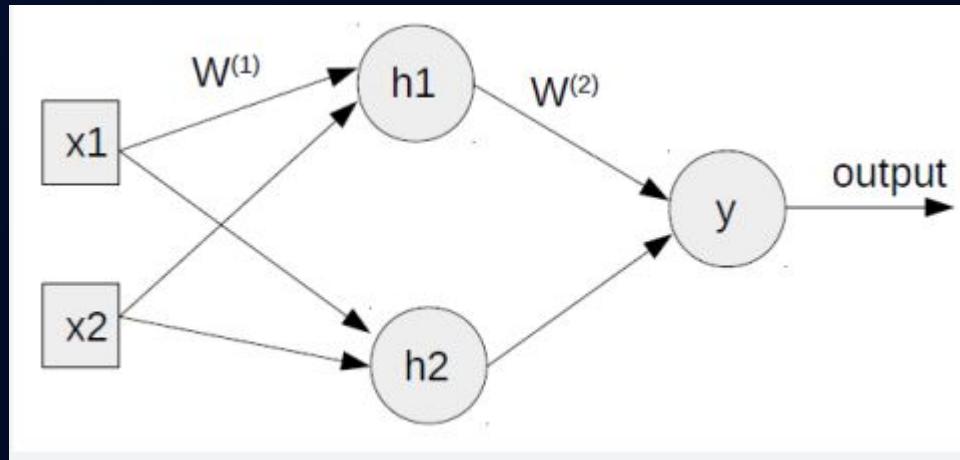
Implementar el algoritmo de perceptrón multicapa y utilizarlo para aprender la Función lógica “O exclusivo” presentada en el Ejercicio 1 (mismos datos y misma salida esperada).



>>> Estructura

- Planteamos 2 nodos iniciales
- Planteamos 1 capa intermedia:
 - TanhLayer que tiene 2 nodo y recibe 2 datos
- Plantemos 1 nodo final Tanh
- Learning rate: 0.01
- Epochs: 20000
- Error_min: 0.02
- Los betas en 0

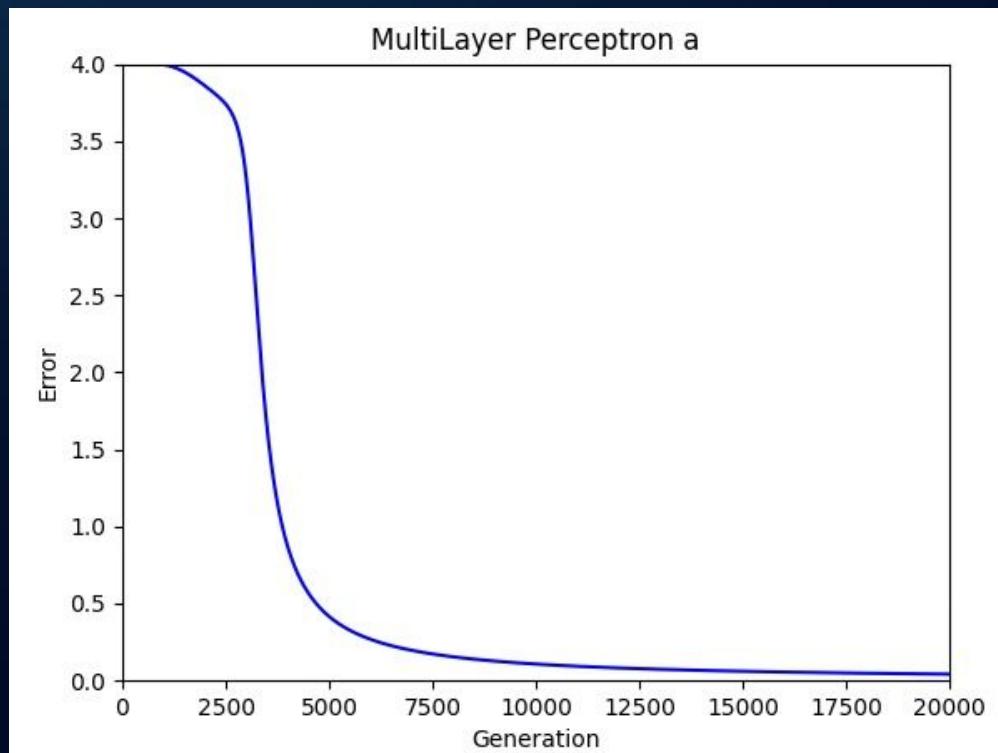
```
input=0output= 0.0072
expected output= [0]
input=1output= 0.0065
expected output= [0]
input=2output= 0.9876
expected output= [1]
input=3output= 0.9876
expected output= [1]
error entrenamiento=0.03840734488787733
```



Resultados: Curva de errores

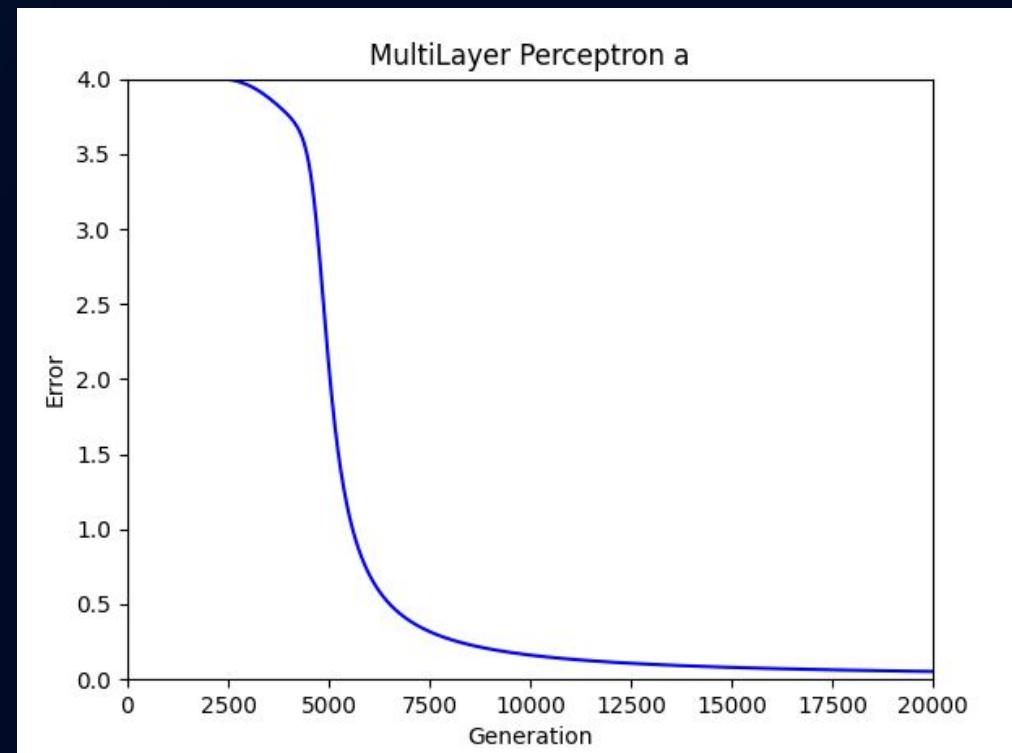
Con MOMENTUM

```
{  
    "learning_rate": 0.01,  
    "epochs": 20000,  
    "min_error": 0.02,  
    "optimisation_method": "momentum",  
    "momentum": 0.2  
}
```



sin MOMENTUM

```
{  
    "learning_rate": 0.01,  
    "epochs": 20000,  
    "min_error": 0.02,  
    "optimisation_method": "NONE",  
    "momentum": 0.2  
}
```



»»» ¿Cómo medimos la performance de los perceptrones en el punto B y C?

Con el accuracy



Ejercicio 3 B

Consigna

Implementar el algoritmo de perceptrón multicapa y utilizarlo para aprender a discriminar si un número es “par”, con entradas dadas por el conjunto de números decimales del 0 al 9 en el archivo “TP2-ej3-digitos.txt”.



¿Cómo se evalúa el accuracy en este punto?

El perceptrón da como salida valores en el rango [0 , 1]

Se toma como una salida acertada si el resultado esperado era par y el obtenido fue mayor a 0.5, y por el contrario, si el valor esperado era impar y el obtenido fue menor a 0.5

Luego se divide la suma de los resultados acertados por el tamaño del conjunto de testeo.

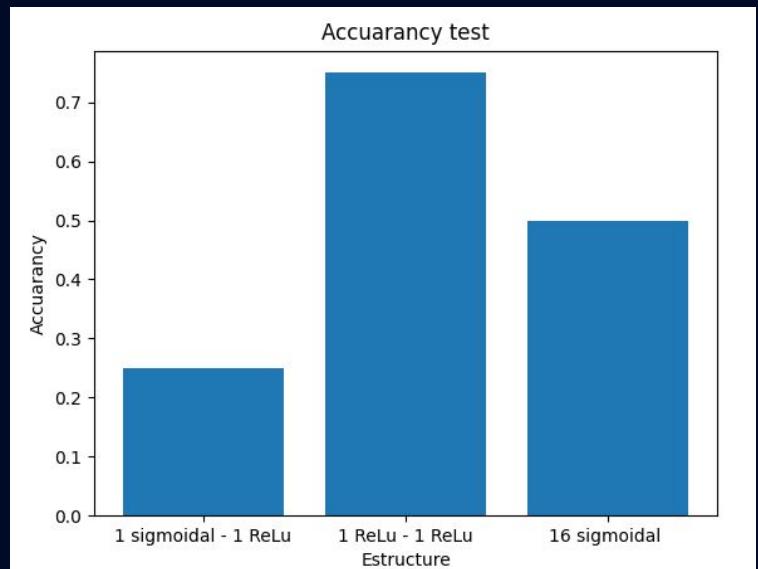
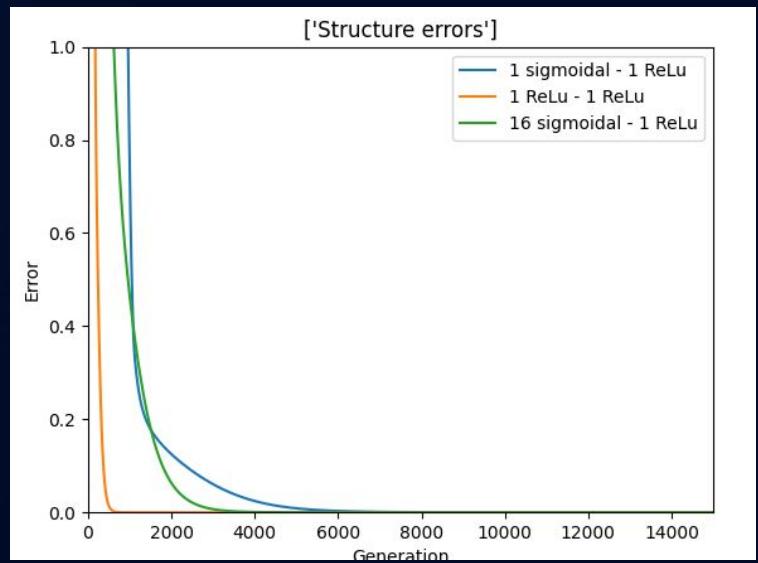


» Que estructuras probamos

- Planteamos 1 capa intermedia:
 - Sigmoidal que tiene 1 neurona y recibe 35 datos
- Plantemos 1 nodo final ReLu
- Planteamos 1 capa intermedia:
 - ReLu que tiene 1 neurona y recibe 35 datos
- Plantemos 1 nodo final ReLu
- Planteamos 1 capa intermedia:
 - Sigmoidal que tiene 16 neurona y recibe 35 datos
- Plantemos 1 nodo final ReLu

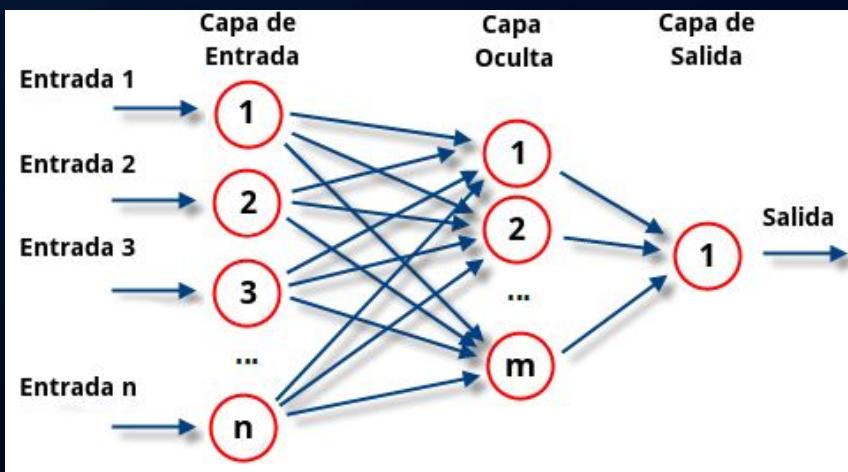
```
{  
  "learning_rate": 0.01,  
  "epochs": 15000,  
  "min_error": 0.02,  
  "optimisation_method": "NONE",  
  "momentum": 0.2  
}
```

Se puede ver que el que tiene valores correctos es la tercer estructura



»»» Estructura Final

- Planteamos 35 nodos iniciales
- Planteamos 1 capa intermedia:
 - Sigmoidal que tiene 16 neuronas y recibe 35 datos
- Plantemos 1 nodo final ReLu
- Learning rate: 0.01
- Epochs: 15000
- Error_min: 0.02
- Los betas en 0



```
input=0output= 1.0000
expected output= [1]
input=1output= 0.0000
expected output= [0]
input=2output= 1.0000
expected output= [1]
input=3output= 0.0000
expected output= [0]
input=4output= 1.0000
expected output= [1]
input=5output= 0.0000
expected output= [0]
input=6output= 1.0000
expected output= [1]
input=7output= 0.0000
expected output= [0]
error entrenamiento=1.9243208693175973e-08
TEST
input=0output= 0.5338
expected output= [1]
input=1output= 0.1671
expected output= [0]
error test=0.6333162052722916
```

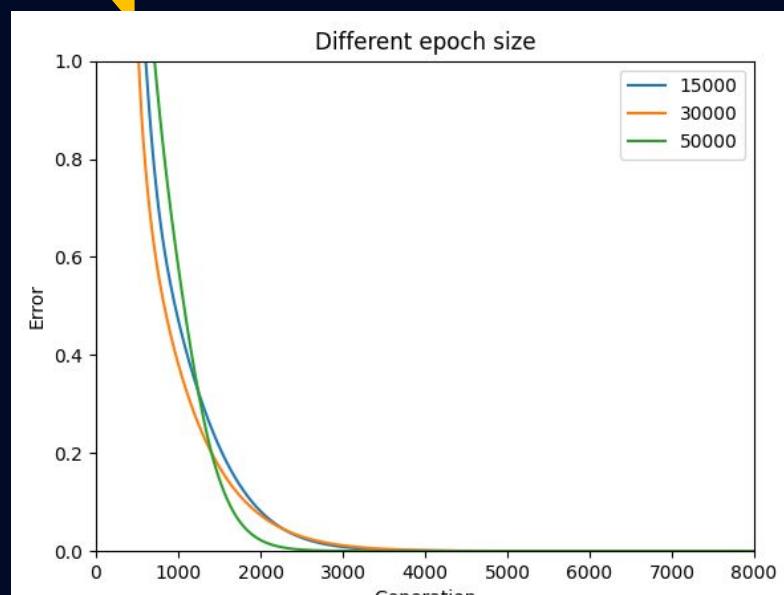
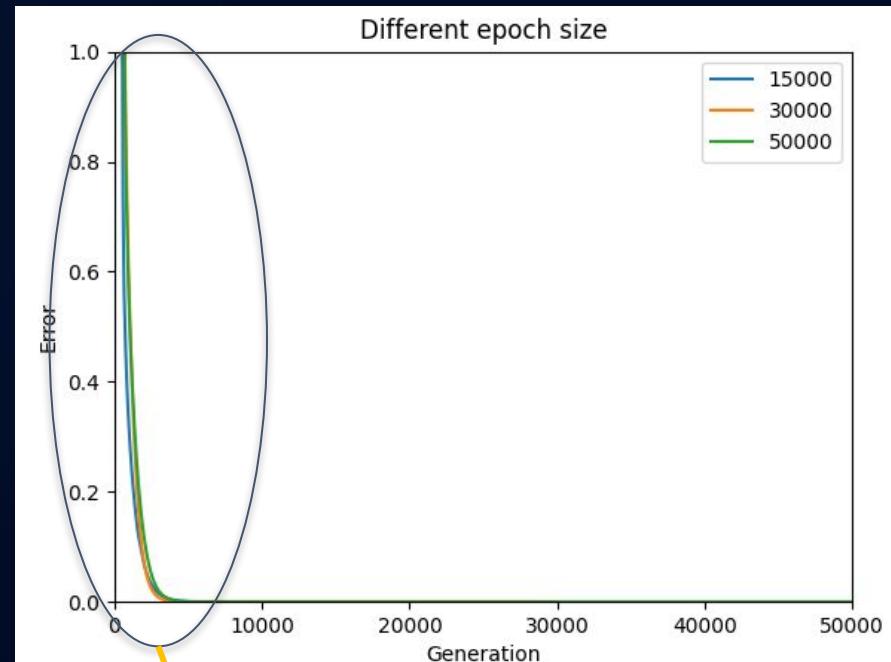
» Que configuración probamos con la estructura elegida

```
{  
  "learning_rate": 0.01,  
  "epochs": 15000,  
  "min_error": 0.02,  
  "optimisation_method": "NONE",  
  "momentum": 0.2  
}
```

```
{  
  "learning_rate": 0.01,  
  "epochs": 50000,  
  "min_error": 0.02,  
  "optimisation_method": "NONE",  
  "momentum": 0.2  
}
```

```
{  
  "learning_rate": 0.01,  
  "epochs": 30000,  
  "min_error": 0.02,  
  "optimisation_method": "NONE",  
  "momentum": 0.2  
}
```

Se puede ver que el error se achica antes del 10000, por lo que tomar mas que eso no tiene sentido



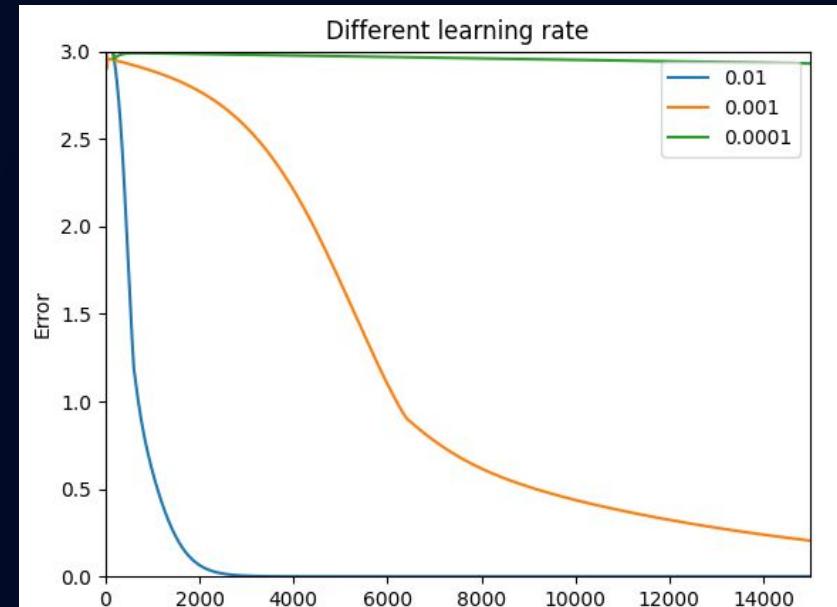
» Que configuración probamos con la estructura elegida

```
{  
  "learning_rate": 0.01,  
  "epochs": 15000,  
  "min_error": 0.02,  
  "optimisation_method": "NONE",  
  "momentum": 0.2  
}
```

```
{  
  "learning_rate": 0.0001,  
  "epochs": 15000,  
  "min_error": 0.02,  
  "optimisation_method": "NONE",  
  "momentum": 0.2  
}
```

```
{  
  "learning_rate": 0.001,  
  "epochs": 15000,  
  "min_error": 0.02,  
  "optimisation_method": "NONE",  
  "momentum": 0.2  
}
```

Viendo estos casos,
elegimos usar learning
rate: 0.01 , ya que el
resto no son
consistentes

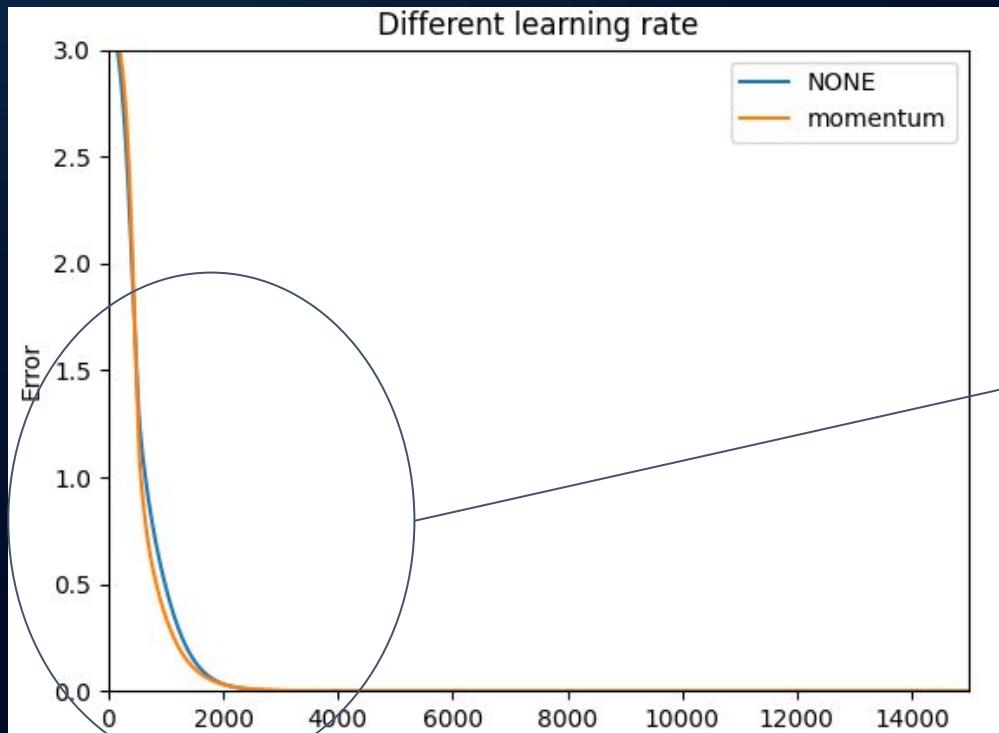


Resultados: Curva de errores

Con MOMENTUM

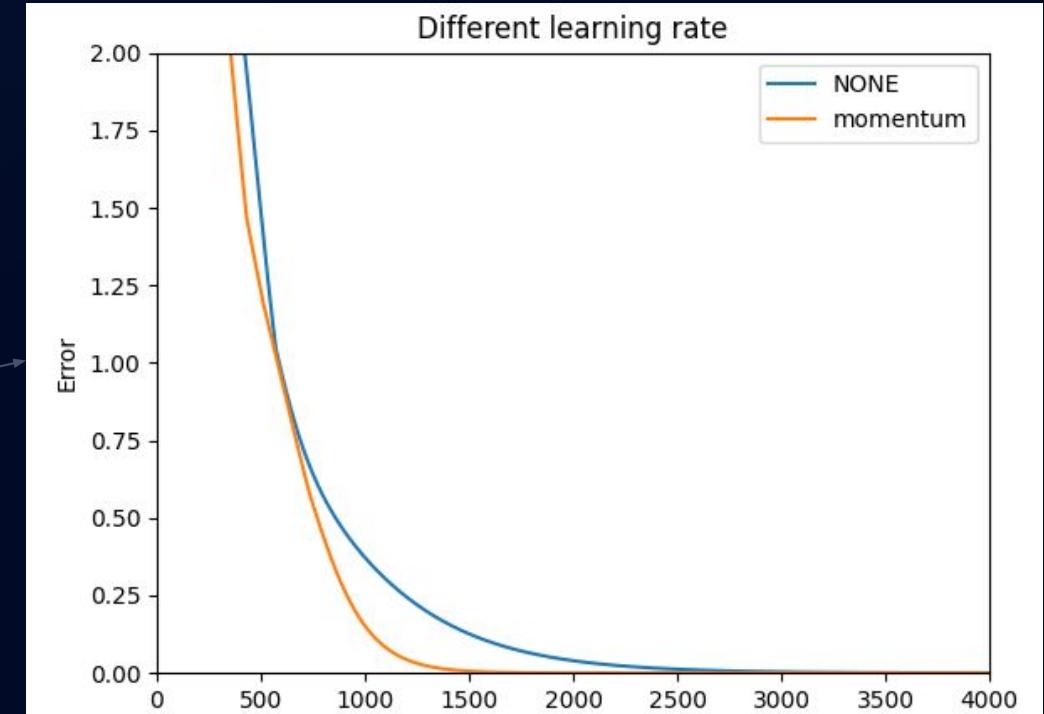
Accuracy = 0.5

Learning rate: 0.01
Generacion: 15000
Min_error: 0.02
Momentum: 0.2



sin MOMENTUM

Accuracy = 0.5



Ejercicio 3 C

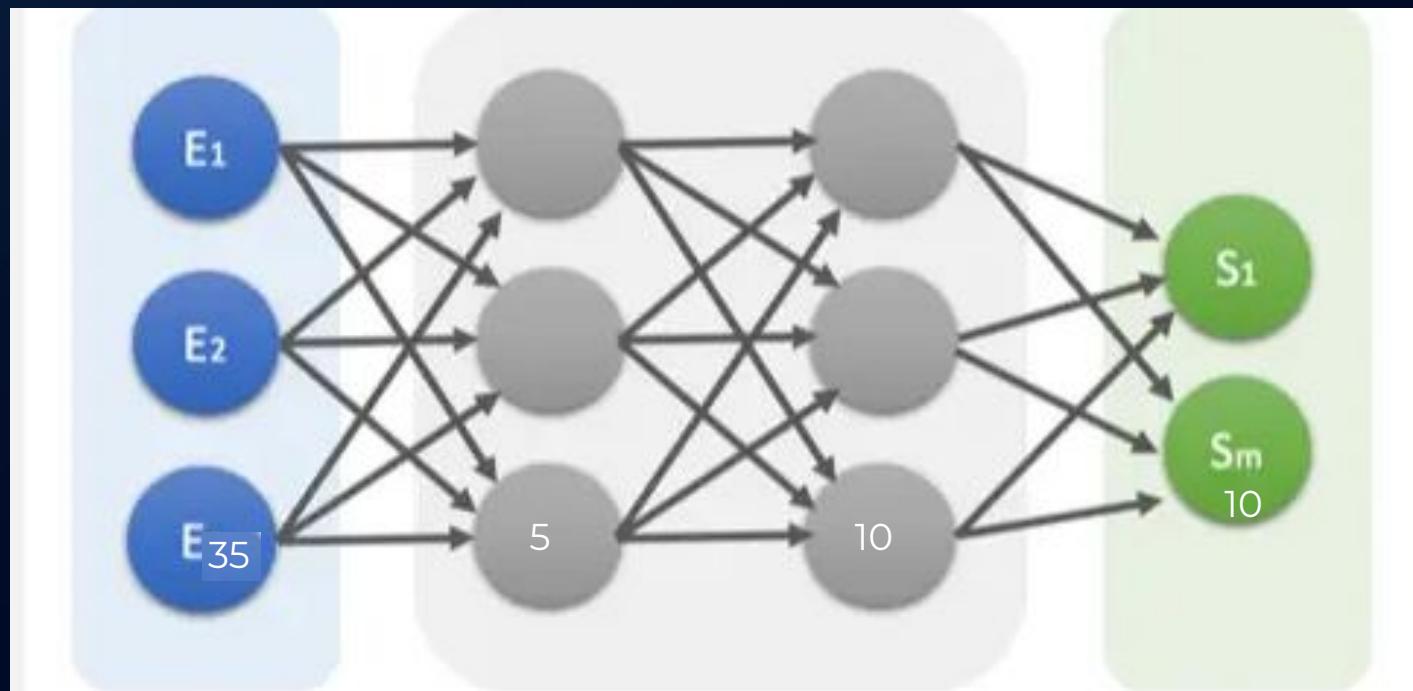
Consigna

Implementar el algoritmo de perceptrón multicapa y utilizarlo para aprender a determinar qué dígito se corresponde con la entrada a la red. Por ejemplo, si alimentamos al perceptrón multicapa con una imagen del dígito “7”, la salida esperada será “7” (la salida puede tomar valores entre 0 y 9). Nótese que se utiliza el mismo perceptrón multicapa, con una salida de 10 neuronas. Una vez que la red haya aprendido, utilizar patrones correspondientes a los dígitos del conjunto de datos, con sus píxeles afectados por ruido. Evaluar los resultados.



»»» Estructura

- Planteamos 35 nodos iniciales
- Planteamos 2 capas intermedia:
 - Sigmoidal que tiene 5 neuronas y recibe 35 datos
 - Sigmoidal que tiene 10 neuronas y reciba 5 datos
- Plantemos 10 nodos finales ReLu
- Learning rate: 0.01
- Epochs: 15000
- Error_min: 0.02
- Los betas en 0



¿Cómo se evalúa el accuracy y error en este punto?

El perceptrón da como salida un array de 10 posiciones, donde cada valor esta en el rango [0, 1]

Se toma como una salida acertada si el indice con mayor valor es el numero esperado como resultado, por ejemplo:

Si el array de salida es:

[0.10 , 0.01, 0.21, 0.11, 0.04, 0.09, 0.12, 0.07, 0.03, 0.91]

Y el valor esperado era: 9

Entonces es una salida acertada

Luego se divide la suma de los resultados acertados por el tamaño del conjunto de testeo

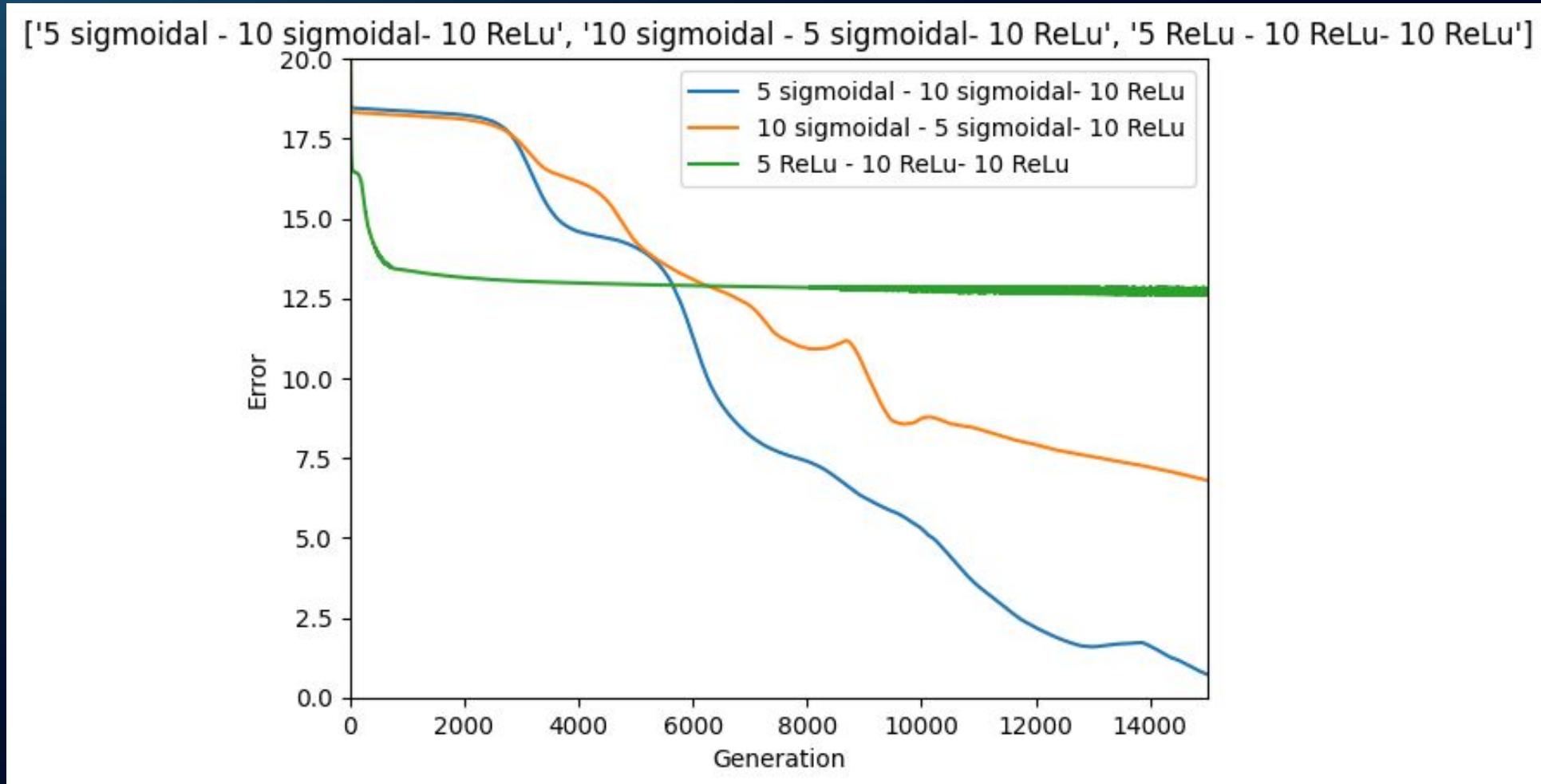
El error en una epoca del entrenamiento se calcula como la suma de las distancias entre el resultado esperado y el obtenido para todos los elementos de entrenamiento.

Si el array de salida es: [0.10 , 0.01, 0.21, 0.11, 0.04, 0.09, 0.12, 0.07, 0.03, 0.91]

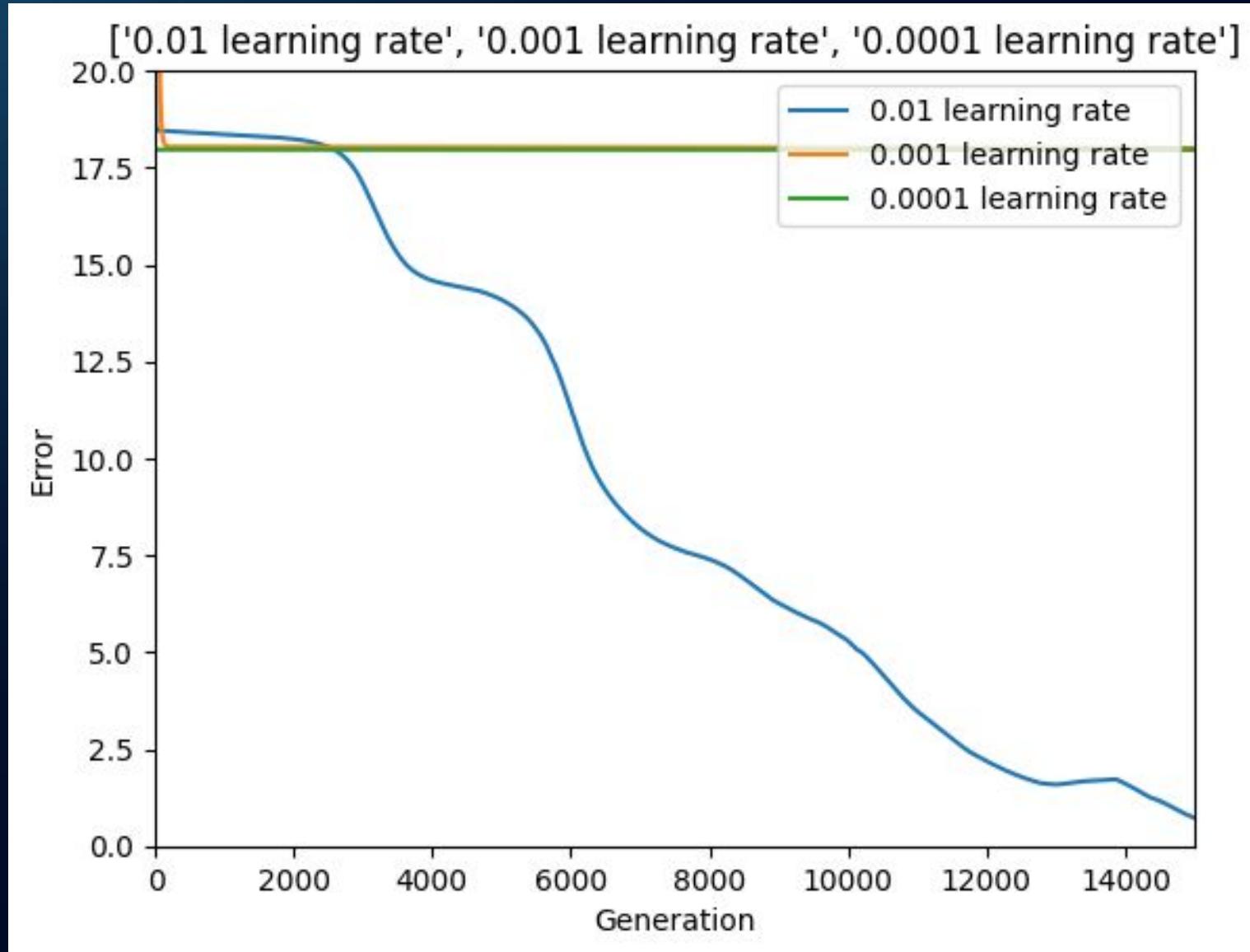
y el valor esperado es 9, en este caso se calcula la distancia entre el array resultado y [0,0,0,0,0,0,1].Se calcula para el resto de los numeros y se los suma.



Resultados: Pruebas de arquitecturas



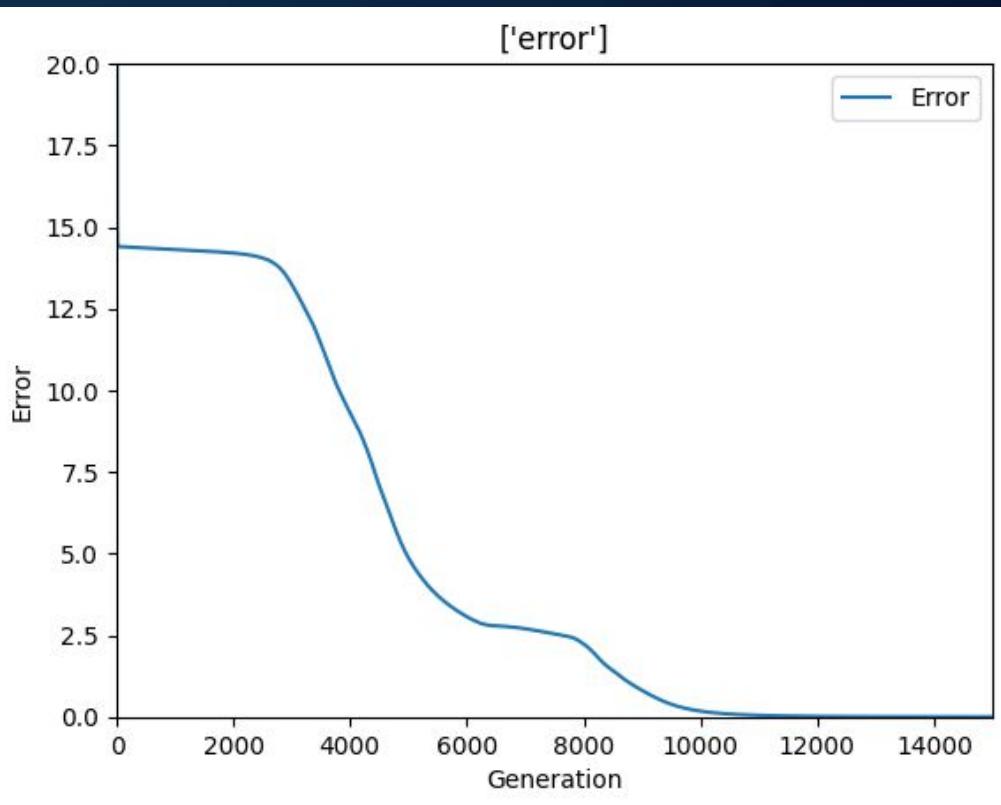
Resultados: Pruebas de learning rate



Resultados: Curva de errores

Con MOMENTUM

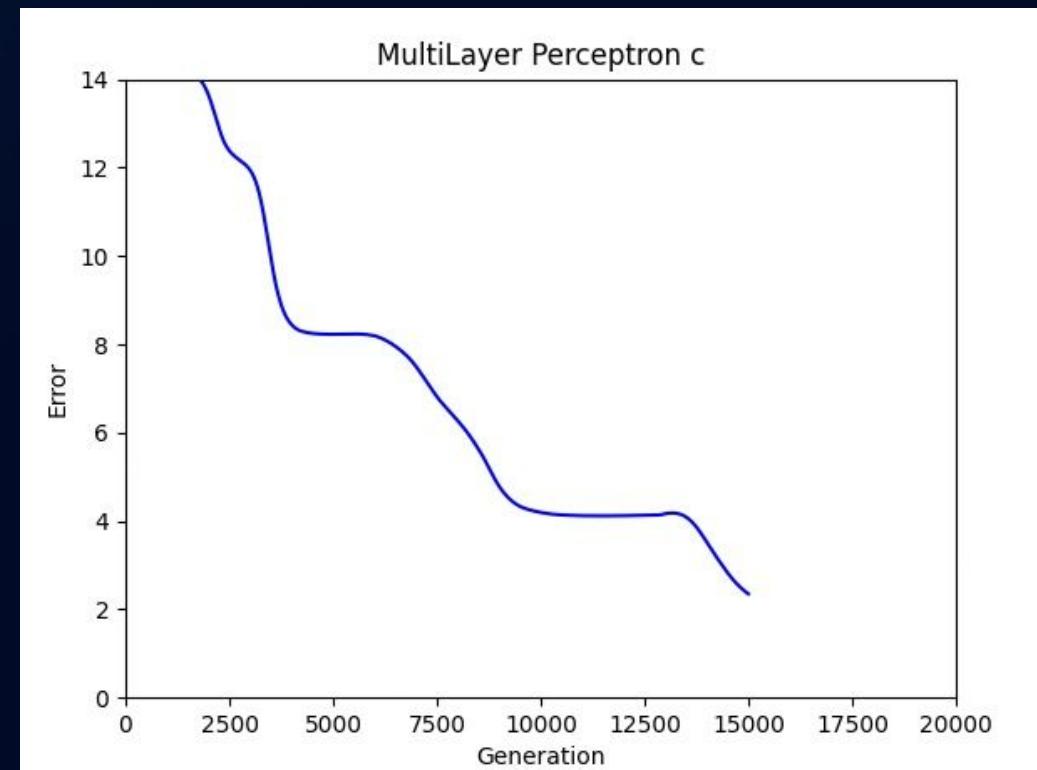
Accuracy = 0.0



Learning rate: 0.01
Generacion:15000
Min_error: 0.02
Momentum:0.2

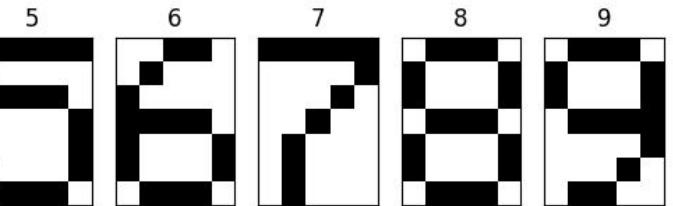
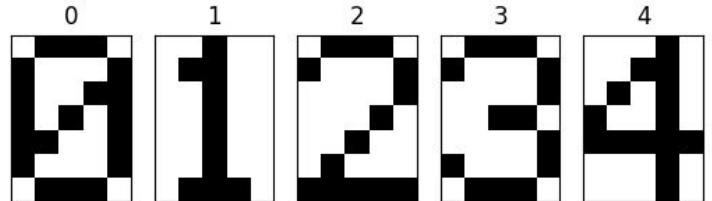
sin MOMENTUM

Accuracy = 0.0

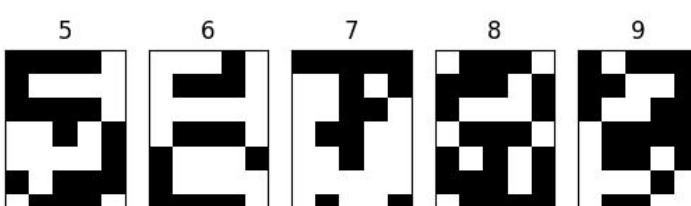
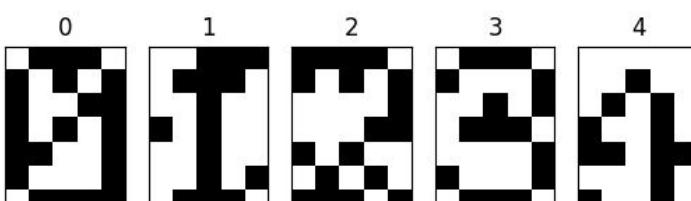


Resultados: Ruido

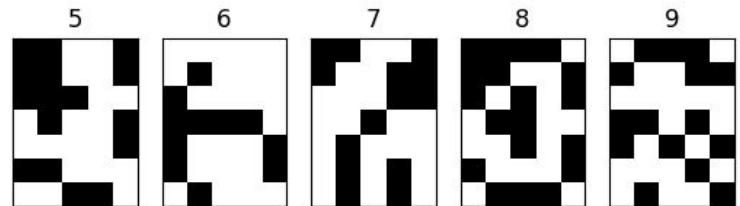
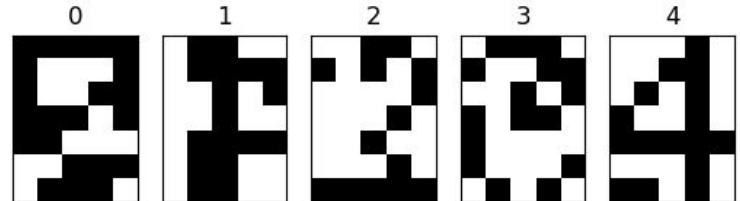
Set de test con ruido=0.0 Linear Accuracy: 1.0



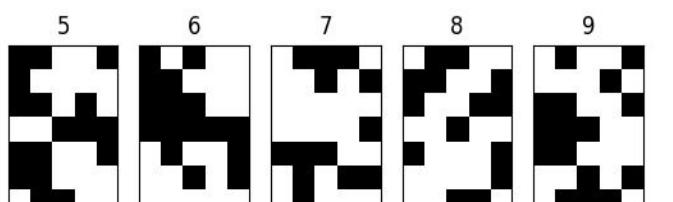
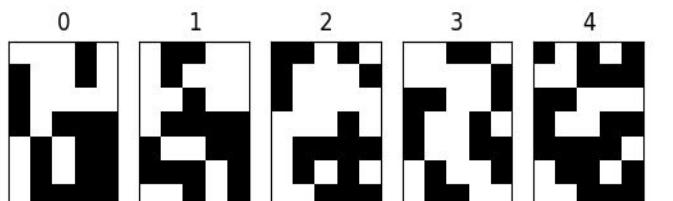
Set de test con ruido=0.1 Linear Accuracy: 0.7



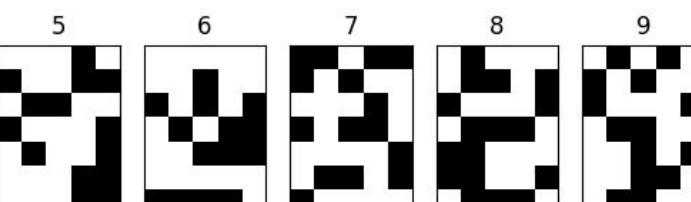
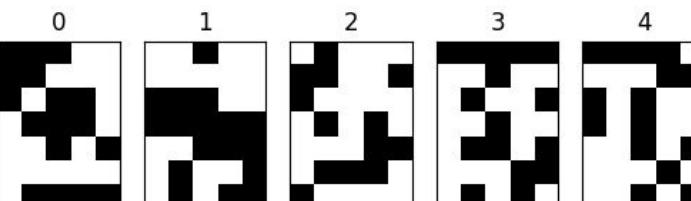
Set de test con ruido=0.2 Linear Accuracy: 0.4



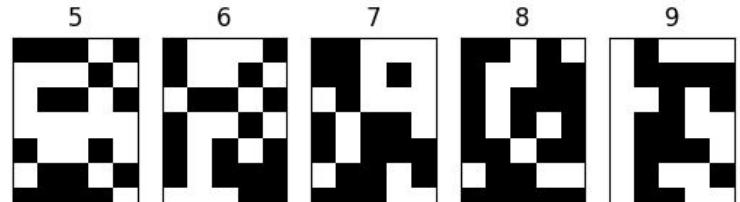
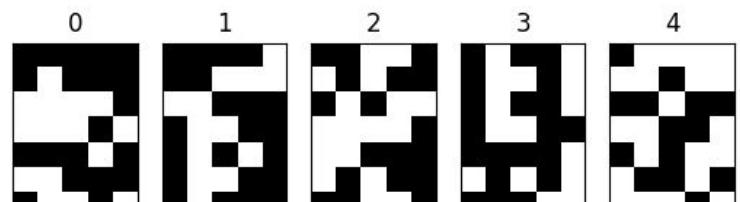
Set de test con ruido=0.30000000000000004 Linear Accuracy: 0.3



Set de test con ruido=0.4 Linear Accuracy: 0.1



Set de test con ruido=0.5 Linear Accuracy: 0.0



»»» Conclusiones



Item A

A diferencia del ejercicio 1, el perceptrón multicapa puede resolver el problema XOR porque tiene múltiples capas ocultas que le permiten aprender representaciones no lineales de los datos, lo que no es posible con un perceptrón simple.



Item B

Se puede reconocer algunas veces si un número es par o no sobre el conjunto de evaluación, exactamente en un 50% de las veces, lo que vuelve este perceptrón una tirada de moneda, donde la mitad de las veces acierta y la mitad no. Por lo tanto, no es útil para resolver esta problemática. Este comportamiento se podría deber a que no hay ningún patrón en la forma de los números que determine si este es par o impar.



Item C

Se puede reconocer perfectamente al conjunto de entrenamiento pero falla con el test de training, esto es porque casi siempre es imposible que pueda diferenciar y poner en la categoría correcta al número (que el número 9 vaya a la categoría 9 por ejemplo) ya que como ningún elemento en el conjunto entraba en la categoría 9, no pudo entrenar esos caminos neuronales.

En cuanto al ruido, vemos que si hay pérdida de información, el perceptrón es afectado a la hora de reconocer las imágenes.



Fin.