SIA - TP3

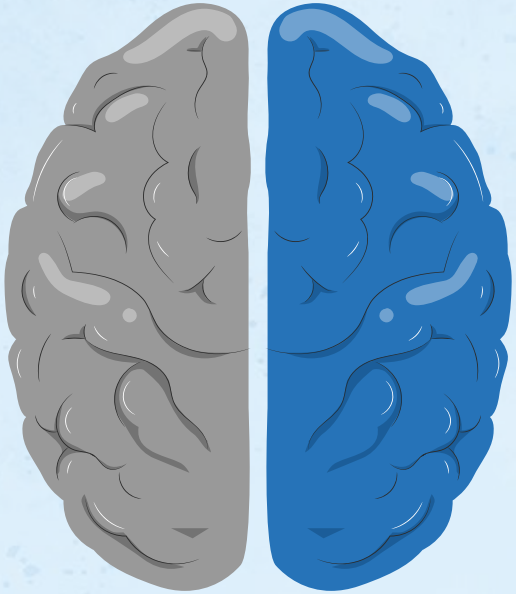
PERCEPTRÓN SIMPLE Y MULTICAPA

Recuperatorio

Grupo 6

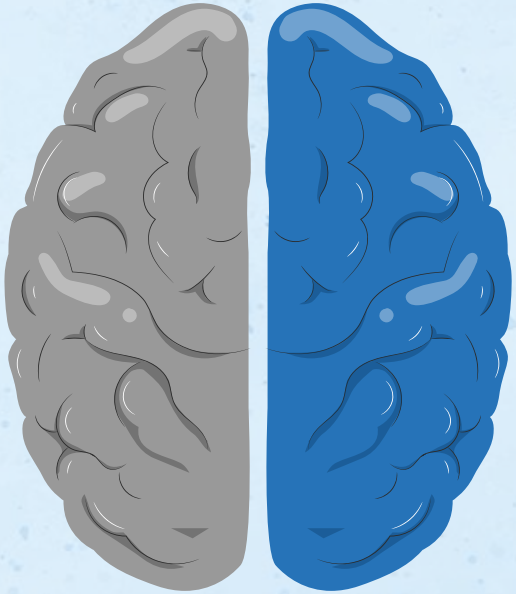
- Francisco Sendot
- Lucia Digon
- Martín E. Zahnd
- Juan Ignacio Fernández Dinardo

Correcciones



Las correcciones principales son:

- Corregir las comparativas de perceptrón lineal/no-lineal
- Agregar variación de arquitecturas en las pruebas, mostrar más detalles importantes que se pueden analizar del perceptrón multicapa
- Agregar análisis del conjunto de datos de *MNIST*



01

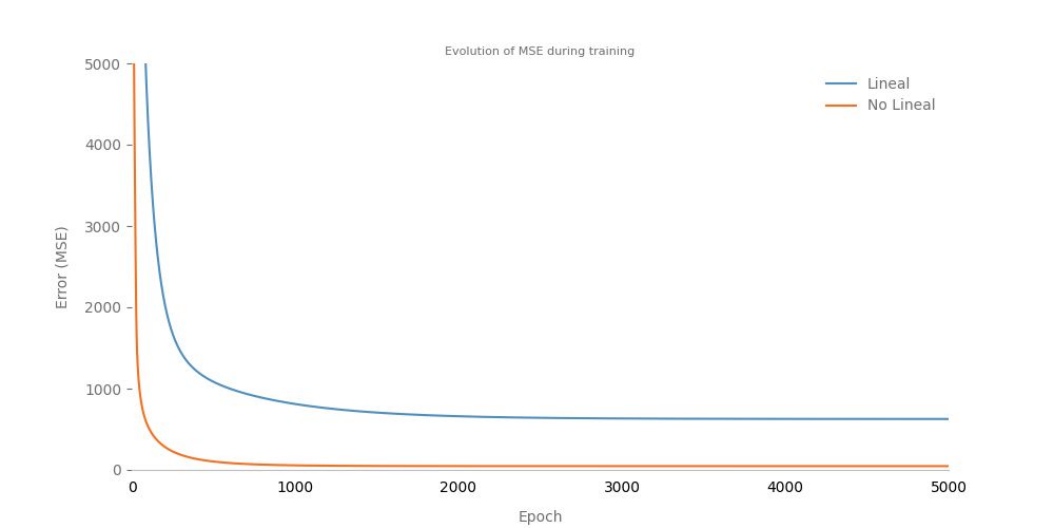
PERCEPTRÓN LINEAL / NO LINEAL

LINEAL VS NO LINEAL (tanh)

Train proportion: 0.7
Learning rate: 0.01
Epochs: 10000

Linear Perceptron
10000/10000
error=626.1914023855614

No Linear
10000/10000
error=46.59498115860013



Los datos parecen seguir un patrón no lineal

VALIDACIÓN CRUZADA: 4-FOLDS

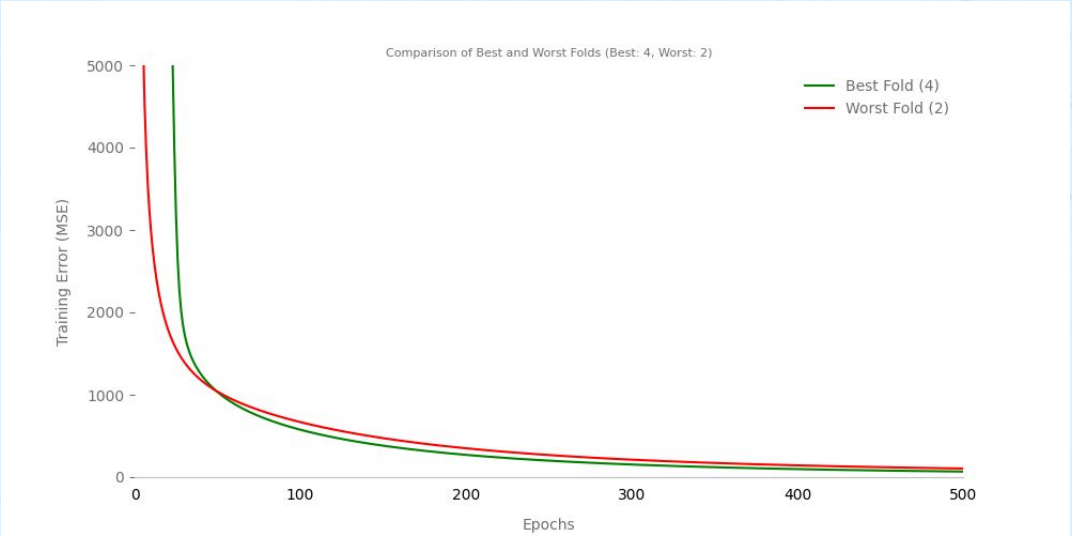
NO LINEAL (tanh)

Train proportion: 0.7
Learning rate: 0.01
Epochs: 10000

Criterio: Error final

===== Fold 4 =====
error=22.381658391274215

===== Fold 2 =====
error=43.88740966303832



VALIDACIÓN CRUZADA: 4-FOLDS

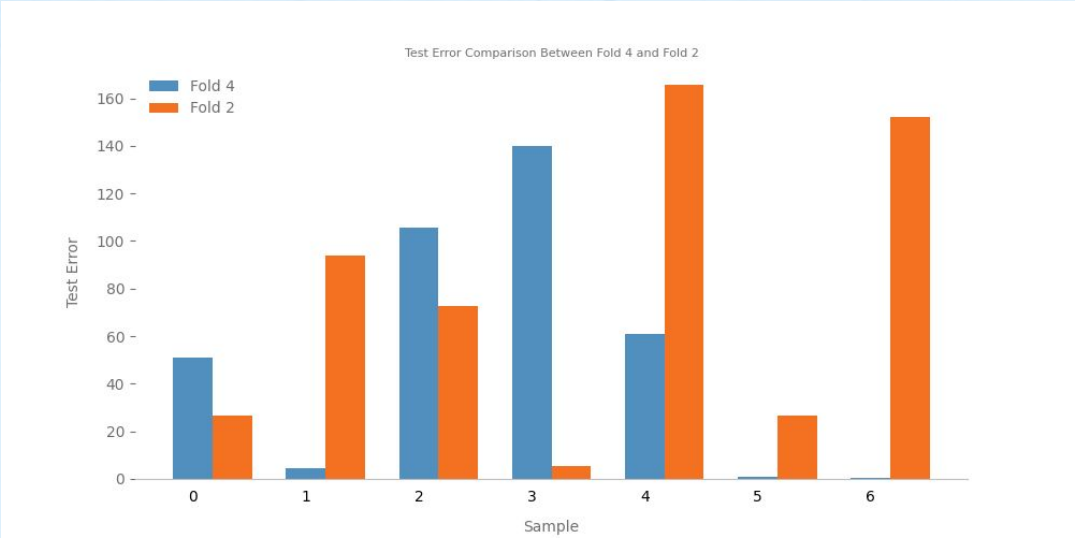
NO LINEAL (tanh)

Veamos que tal evaluan

Train proportion: 0.7
Learning rate: 0.01
Epochs: 10000

===== Fold 4 =====
Error promedio:
51.8892588087337

===== Fold 2 =====
Error promedio:
77.60126208260101



VALIDACIÓN CRUZADA: 4-FOLDS

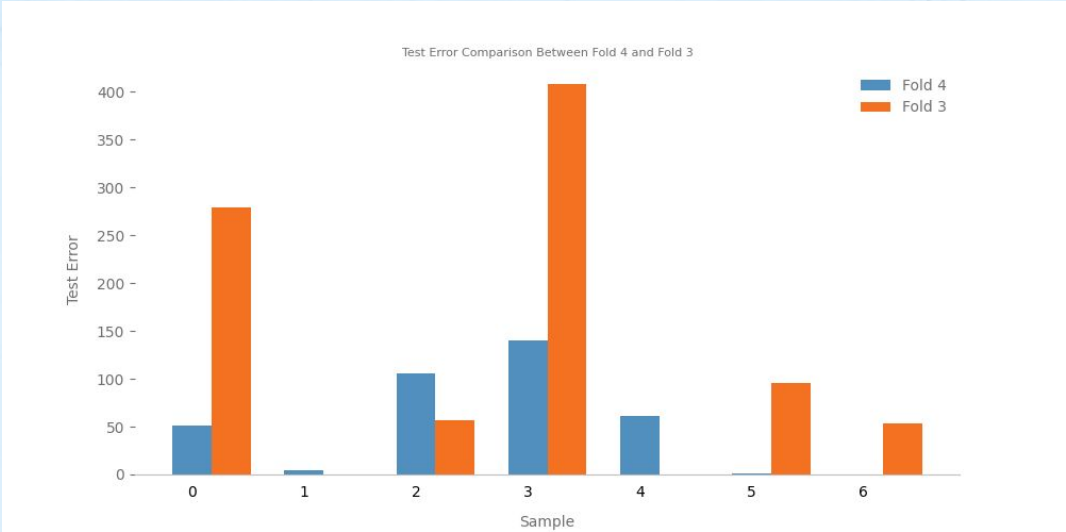
NO LINEAL (tanh)

Que fold evalua mejor? Y cual peor?

Train proportion: 0.7
Learning rate: 0.01
Epochs: 10000

===== Fold 4 =====
Error promedio:
51.8892588087337

===== Fold 3 =====
Error promedio:
127.4068197977374



VALIDACIÓN CRUZADA: 4-FOLDS

NO LINEAL (tanh)

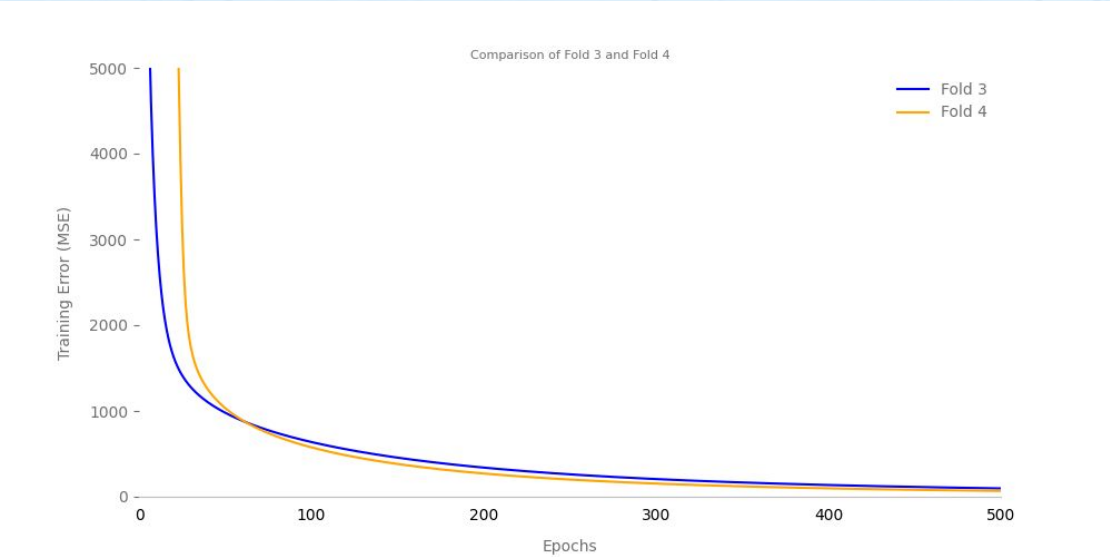
Veamos que tal entrenaron

Train proportion: 0.7
Learning rate: 0.01
Epochs: 10000

===== Fold 4 =====
error=22.381658391274215

===== Fold 3 =====
error=23.272899060086193

Da apenas peor que el “mejor fold” !



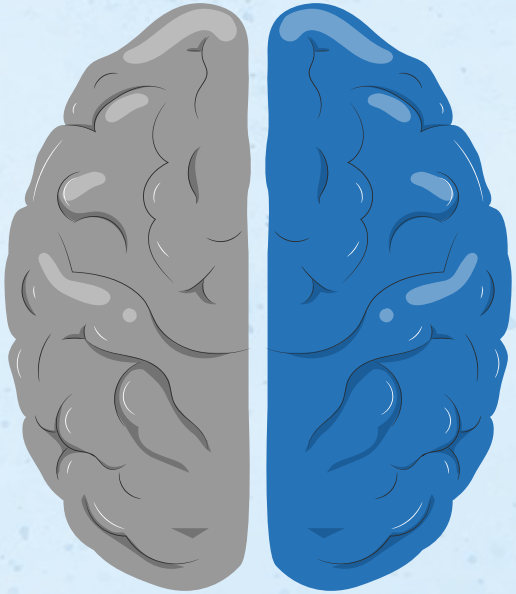
CONCLUSIONES

Entonces?

Mejor capacidad de entrenamiento **no implica estrictamente** mejor capacidad de generalización.

CONCLUSIONES

- El perceptrón lineal mostró que es incapaz de generalizar y aproximar bien el conjunto de datos de entrada de este caso en particular.
- Mediante la validación cruzada se puede observar que hay conjuntos que resultan ser más representativos que otros, puesto que entrenar a la red con estos, permiten una mejor generalización sobre aquellos del set de test.
- Al particionar directamente sobre el set se corre el riesgo de que los datos pueden estar mal distribuidos y, de esta forma, el modelo puede sobre ajustarse y no generalizar bien.
- Es interesante ver que, a pesar de que se pueda generar un buen aprendizaje con un perceptrón, su desempeño a la hora de ver qué tan bien generaliza varía dependiendo del set de datos que usa.

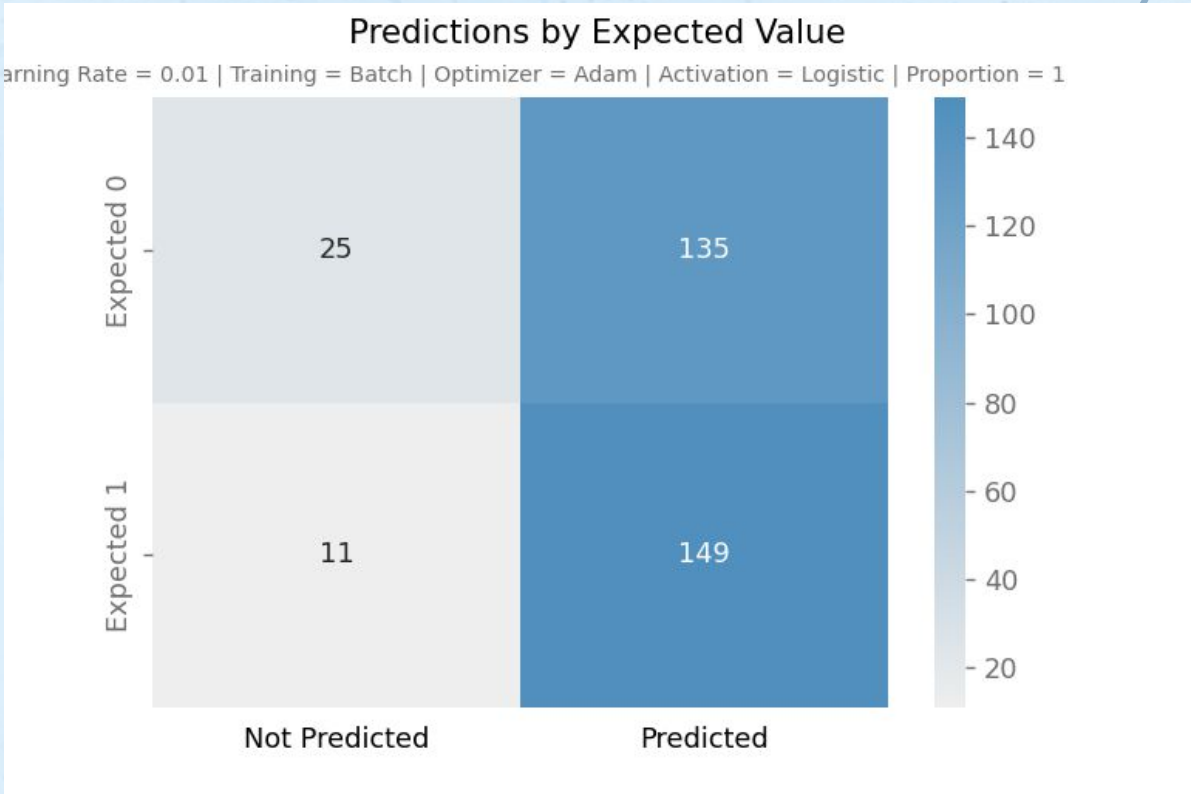


02

PERCEPTRÓN MULTICAPA

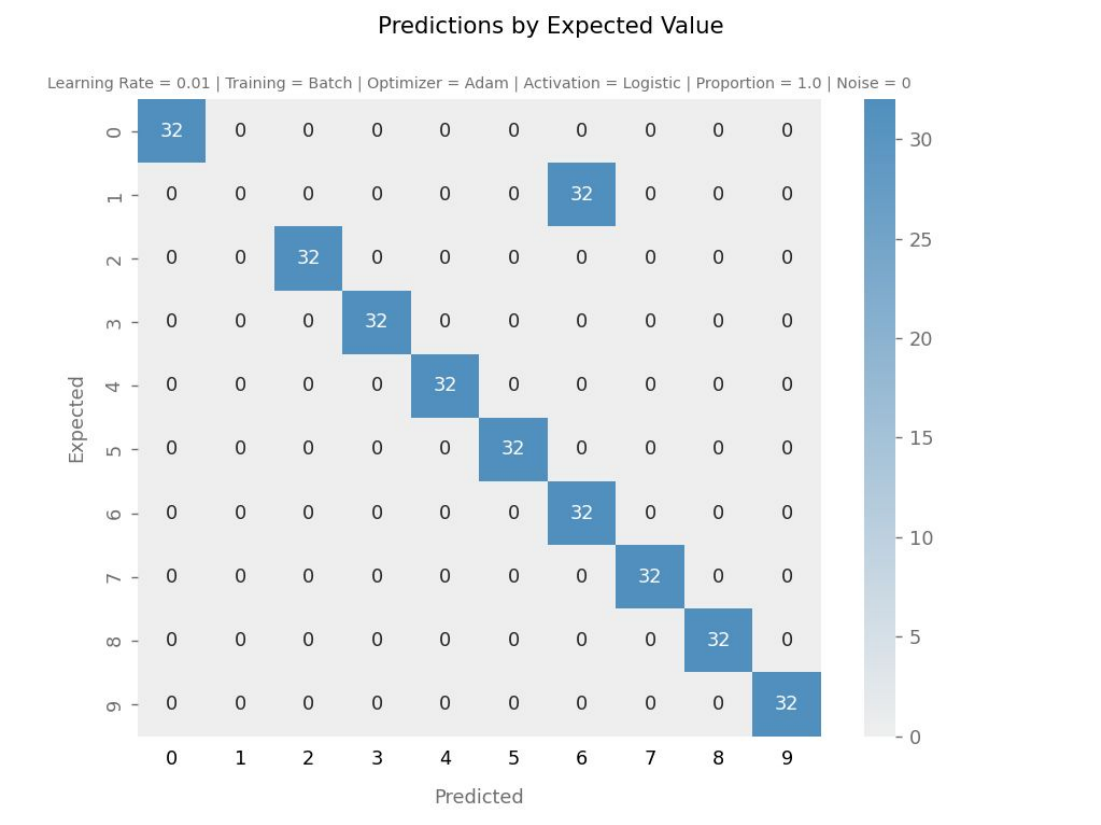
DISCRIMINACIÓN DE PARIDAD

```
epoch = 200
noise = 0.5
beta = 0.4
network = [
     Dense(35, 70, Adam()),
     Logistic(beta),
     Dense(70, 1, Adam()),
     Logistic(beta),

]
```



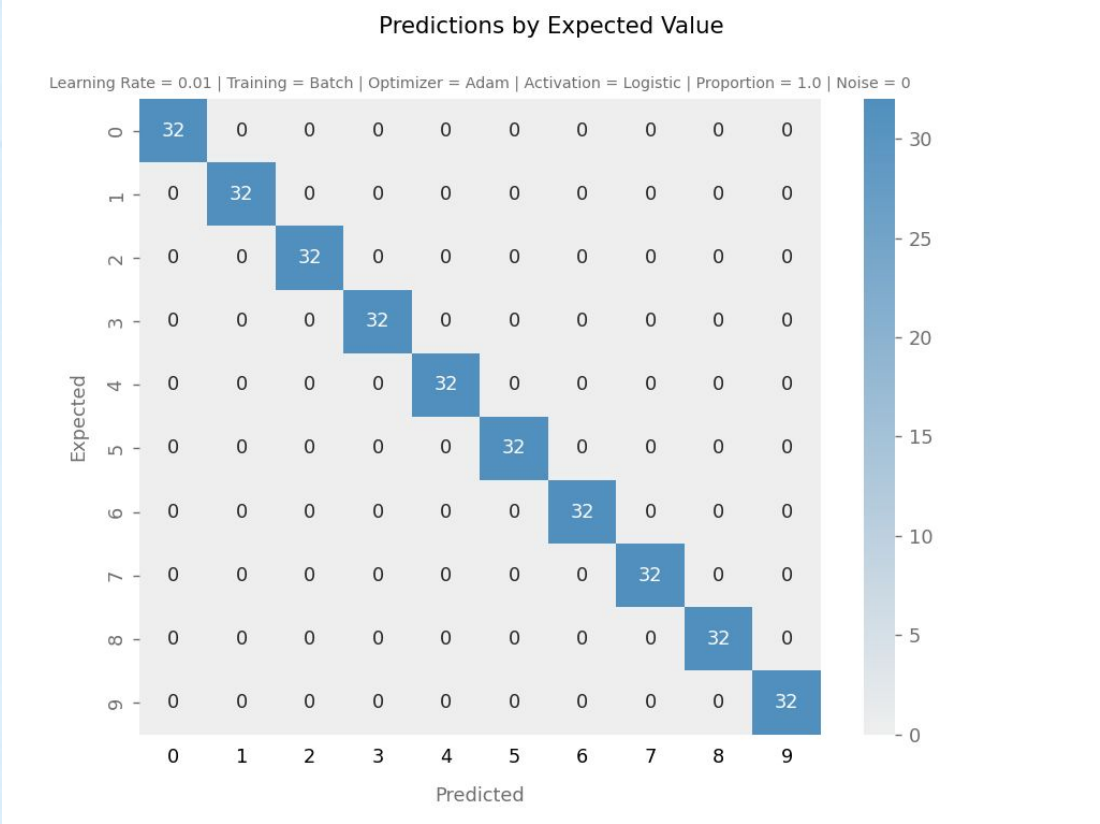
DISCRIMINACIÓN DE DÍGITO

```
epoch = 200
noise = 0
beta = 0.4
network = [
    Dense(35, 70, Adam()),
    Logistic(beta),
    Dense(70, 10, Adam()),
    Logistic(beta),

]
```



DISCRIMINACIÓN DE DÍGITO

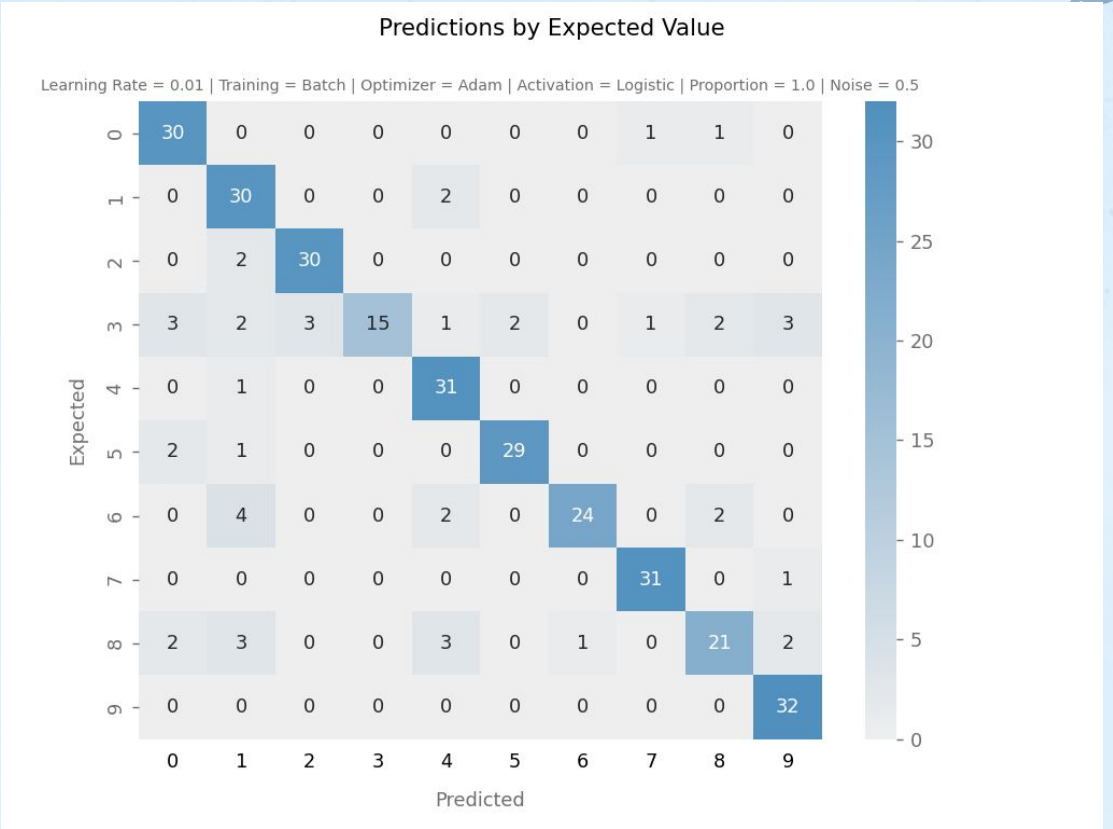
```
epoch = 500
noise = 0
beta = 0.4
network = [
    Dense(35, 70, Adam()),
    Logistic(beta),
    Dense(70, 10, Adam()),
    Logistic(beta),

]
```



DISCRIMINACIÓN DE DÍGITO: CON RUIDO

```
epoch = 500
noise = 0.5
beta = 0.4
network = [
    Dense(35, 70, Adam()),
    Logistic(beta),
    Dense(70, 10, Adam()),
    Logistic(beta),

]
```

Perf = 0,8531



DISCRIMINACIÓN DE DÍGITO

ARQUITECTURA 1:

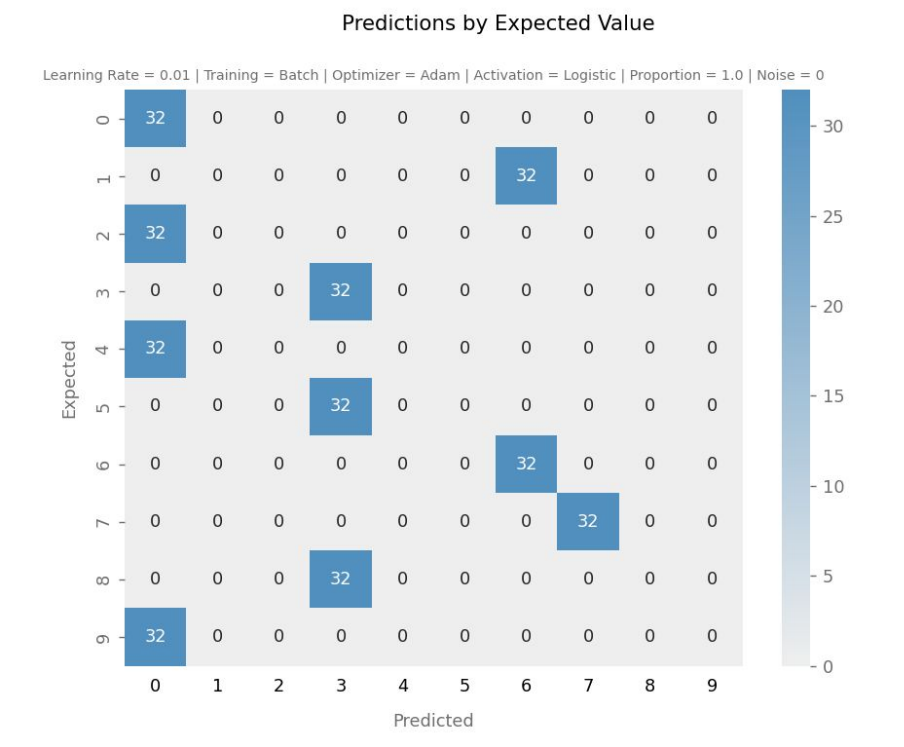
```
epoch = 200
noise = 0
beta = 0.4
network = [
    Dense(35, 70, Adam()),
    Logistic(beta),
    Dense(70, 50, Adam()),
    Logistic(beta),
    Dense(50, 5, Adam()),
    Logistic(beta),
    Dense(5, 10, Adam()),
    Logistic(beta),

]
```

ARQUITECTURA 2:

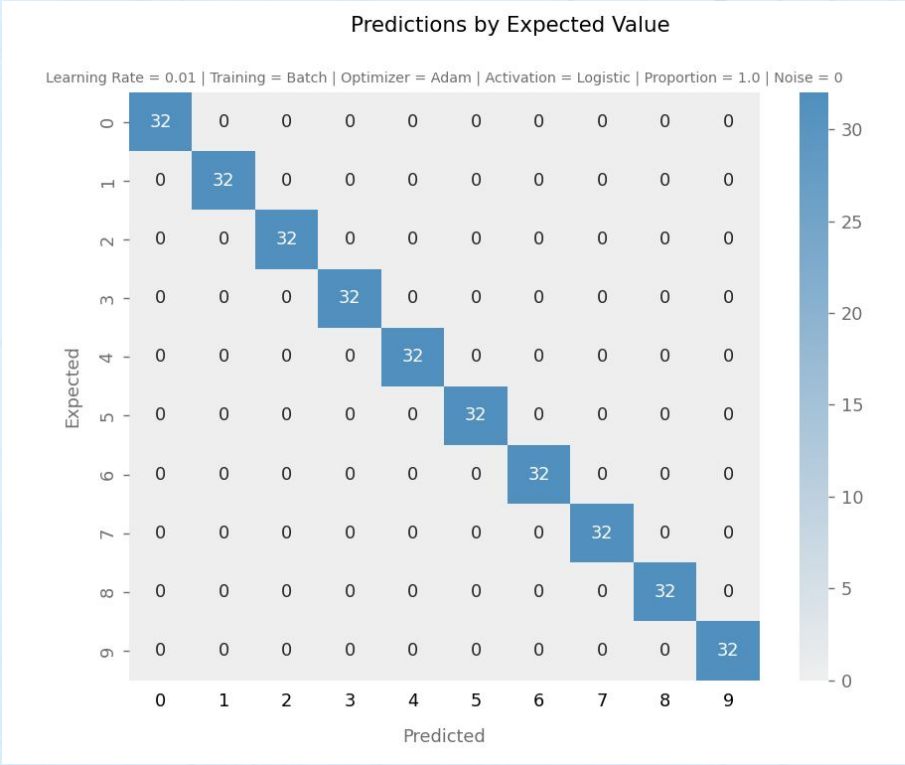
```
epoch = 200
noise = 0
beta = 0.4
network = [
    Dense(35, 70, Adam()),
    Logistic(beta),
    Dense(70, 50, Adam()),
    Logistic(beta),
    Dense(50, 25, Adam()),
    Logistic(beta),
    Dense(25, 10, Adam()),
    Logistic(beta),

]
```

DISCRIMINACIÓN DE DÍGITO

ARQUITECTURA 1:



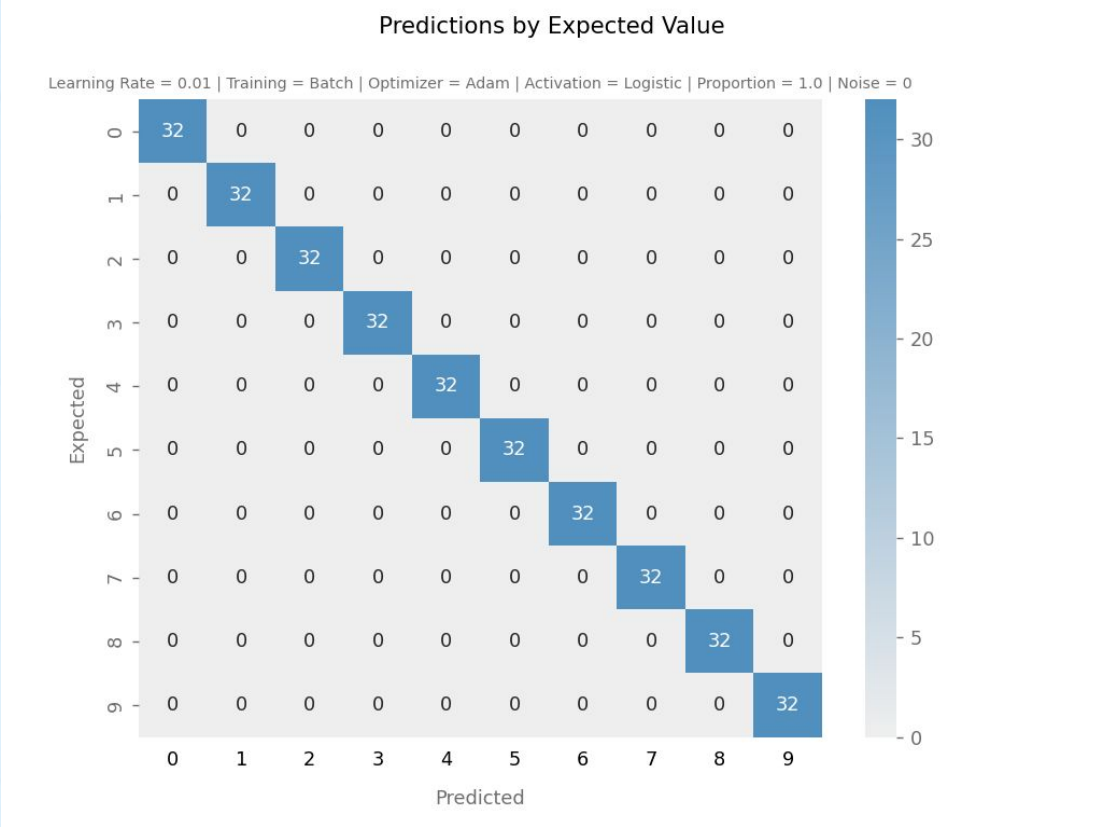
ARQUITECTURA 2:



DISCRIMINACIÓN DE DÍGITO

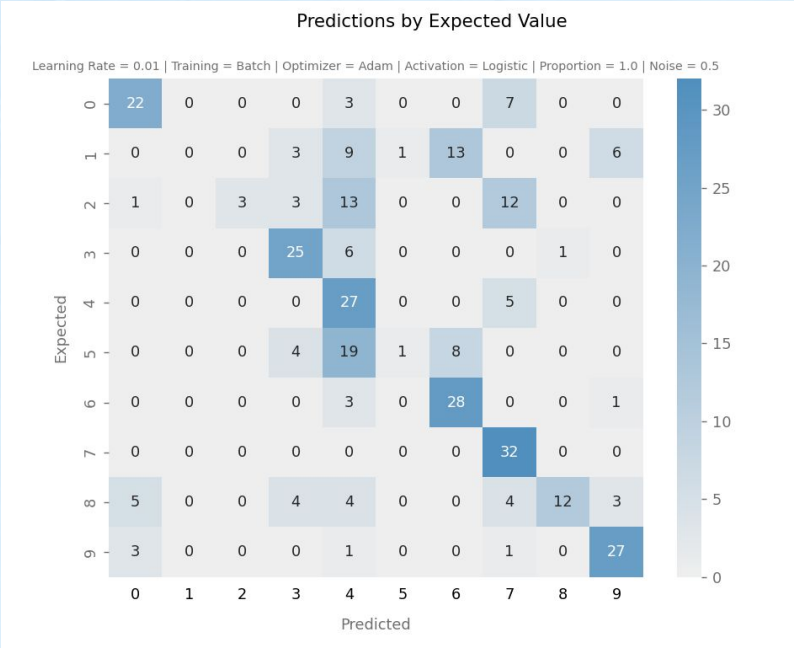
ARQUITECTURA 1:

```
epoch = 700
noise = 0
beta = 0.4
network = [
    Dense(35, 70, Adam()),
    Logistic(beta),
    Dense(70, 50, Adam()),
    Logistic(beta),
    Dense(50, 5, Adam()),
    Logistic(beta),
    Dense(5, 10, Adam()),
    Logistic(beta),

]
```



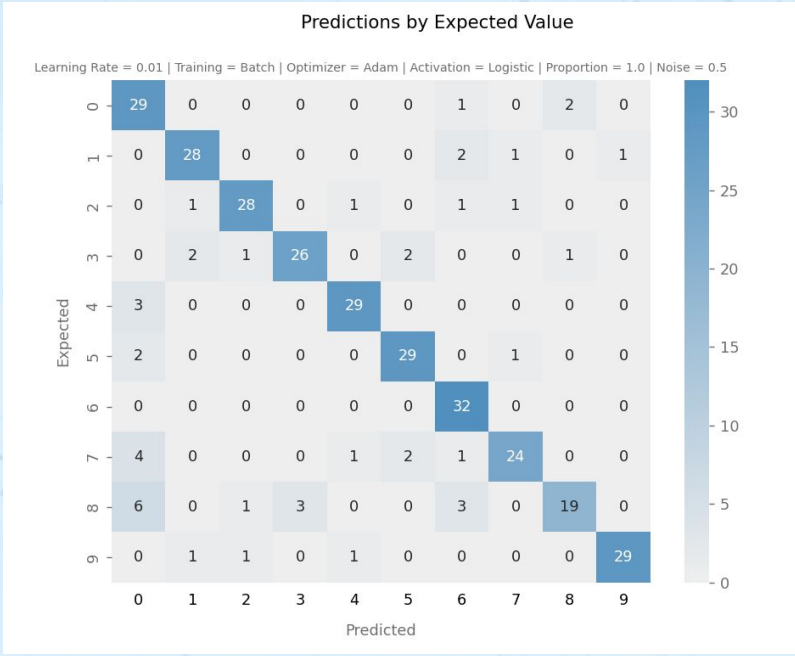
DISCRIMINACIÓN DE DÍGITO: RUIDO 0.5

ARQUITECTURA 1 (700 EPOCHS):



Perf = 0,4917

ARQUITECTURA 2:



Perf = 0,8531

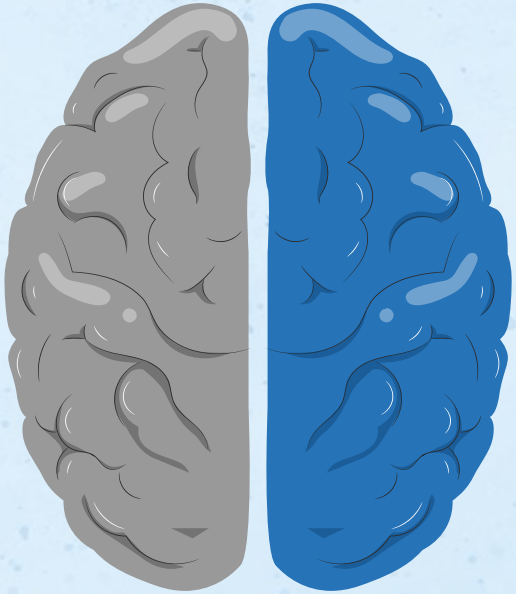
CONCLUSIONES

- Una determinada arquitectura puede llegar a necesitar más epochs para aprender

- Una arquitectura mal diseñada puede generar que, a pesar de que se logre un entrenamiento exitoso, la red no pueda generalizar bien.

- Si se intenta reducir el número de neuronas por debajo de la dimensión del output final, la red puede ser menos eficiente (necesita más *epochs* para aprender) y se puede volver **muy sensible** a los datos ruidosos, es decir, empeora mucho su generalización.

- No hay una regla general para elegir una mejor arquitectura, más capas no necesariamente significa mejores resultados. Sí hay un punto medio que puede ayudar a obtener una mejor generalización.



03

MNIST

DISCRIMINACIÓN DE DÍGITO: MNIST

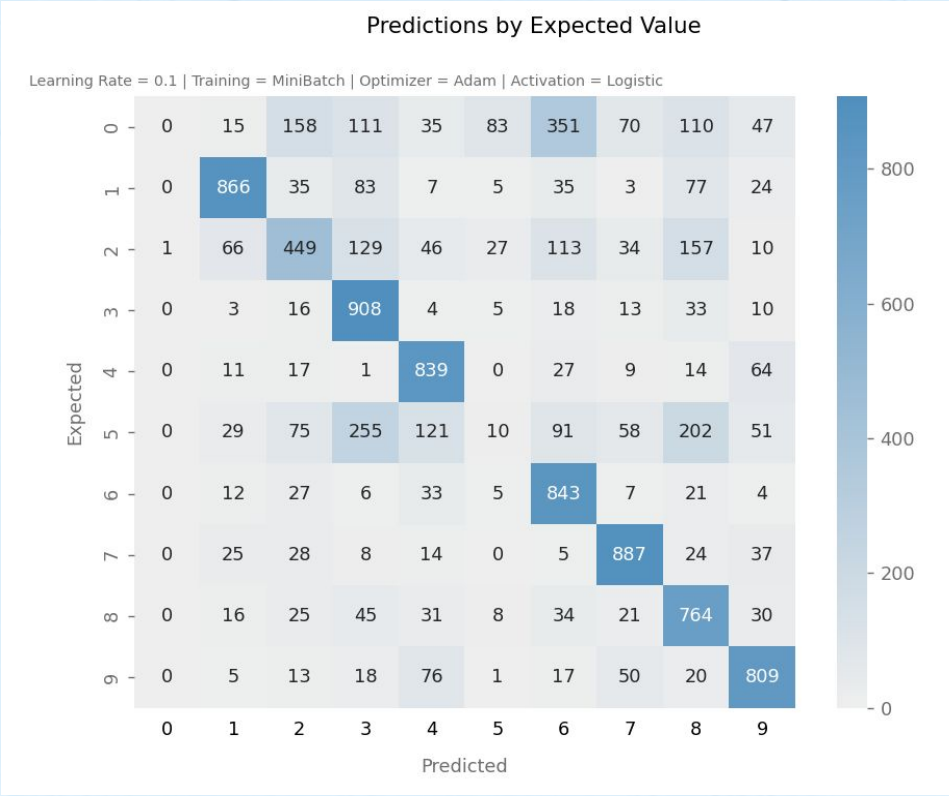
- El Dataset ahora tiene un input y output de 60000 datos
- El input son matrices de 28x28, con cada celda pudiendo tomar un valor entre 0 y 255
- El output se transforma a un vector de probabilidades de que la imagen sea el dígito determinado
- Son **MUCHOS DATOS**
- Se tienen que cambiar decisiones de arquitectura para que pueda correr en tiempo y forma
  - *epoch*=10/100/5/50
  - Utilizamos muchas arquitecturas para lograr encontrar la mejor generalización
- Se usa Mini Batch con un *batch size* de 10000
- Como optimizador se eligió *Adam*
- Como función de activación se eligió *Logistic*
- Tiene un dataset para test (no es necesario agregar ruido)

MNIST

Primer Intento

```
epoch = 10
beta = 0.4
network = [
    Dense(784, 10, Adam()),
    Logistic(beta)
]
```

Perf = 0,6375

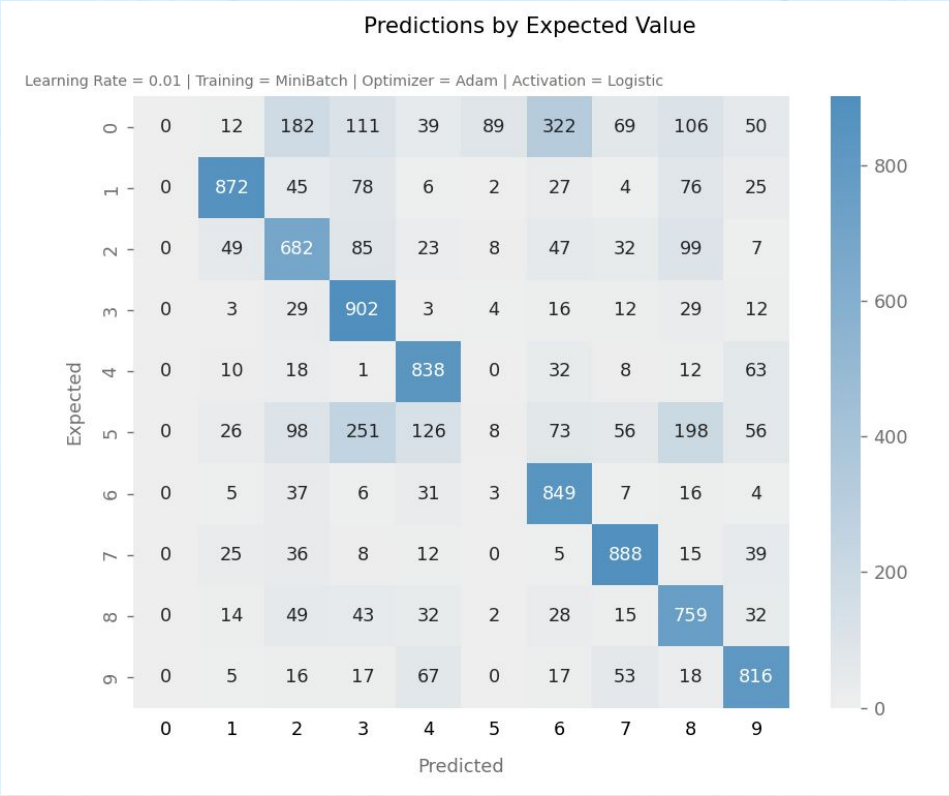


MNIST

Segundo Intento

```
epoch = 100
beta = 0.4
network = [
    Dense(784, 10, Adam()),
    Logistic(beta)
]
```

Perf = 0,6614

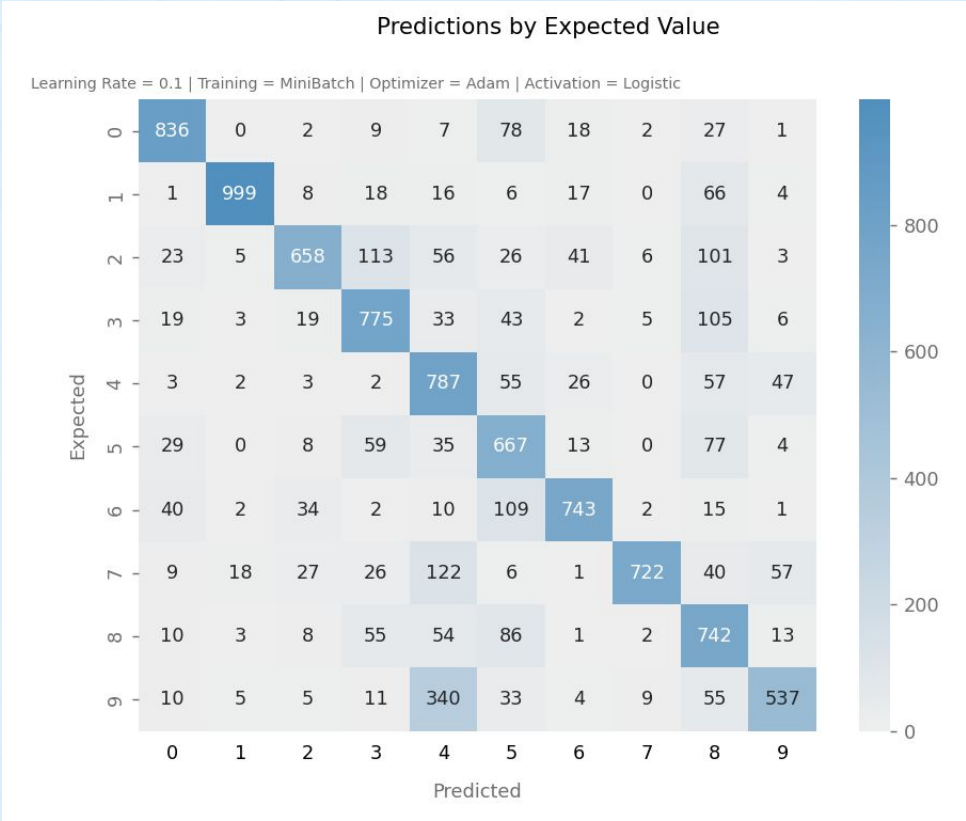


MNIST

Tercer Intento

```
epoch = 5
beta = 0.4
network = [
    Dense(784, 128, Adam()),
    Logistic(beta),
    Dense(128, 64, Adam()),
    Logistic(beta),
    Dense(64, 32, Adam()),
    Logistic(beta),
    Dense(32, 10, Adam()),
    Logistic(beta)
]
```

Perf = 0,7466

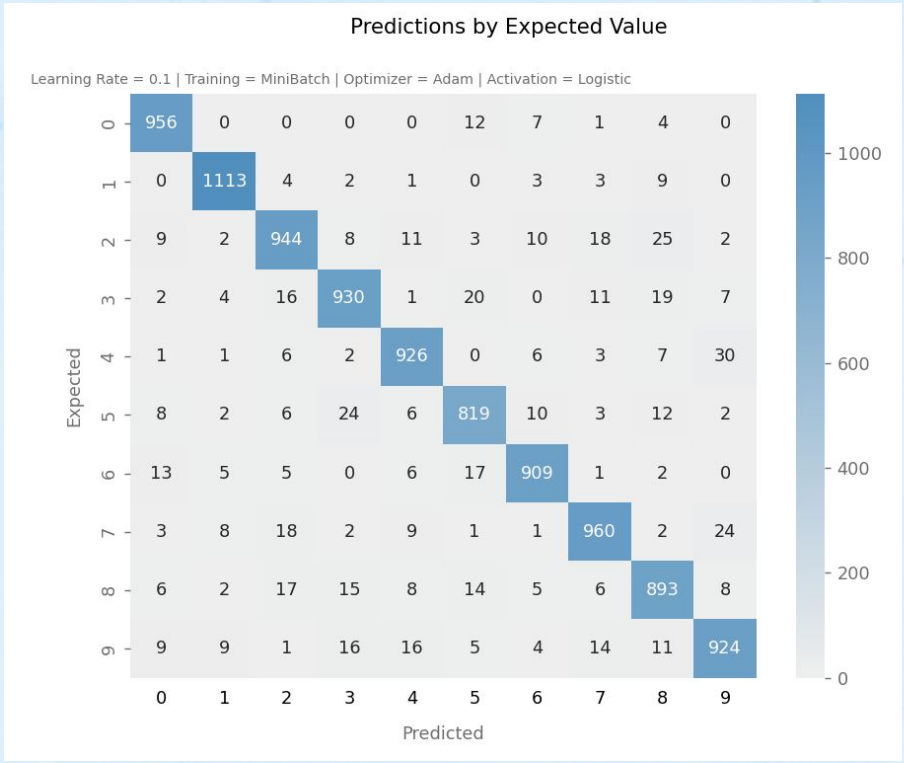


MNIST

Cuarto Intento

```
epoch = 50
beta = 0.4
network = [
    Dense(784, 128, Adam()),
    Logistic(beta),
    Dense(128, 64, Adam()),
    Logistic(beta),
    Dense(64, 32, Adam()),
    Logistic(beta),
    Dense(32, 10, Adam()),
    Logistic(beta)
]
```

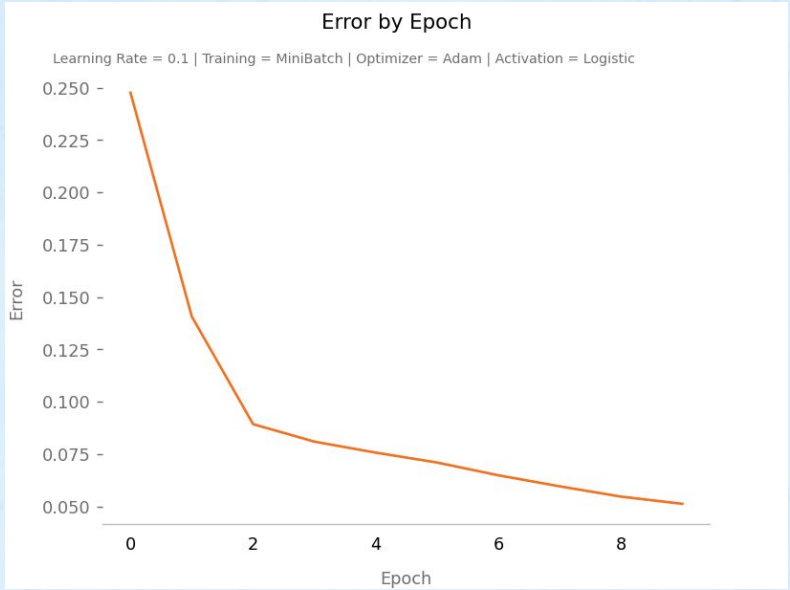
Perf = 0,9374



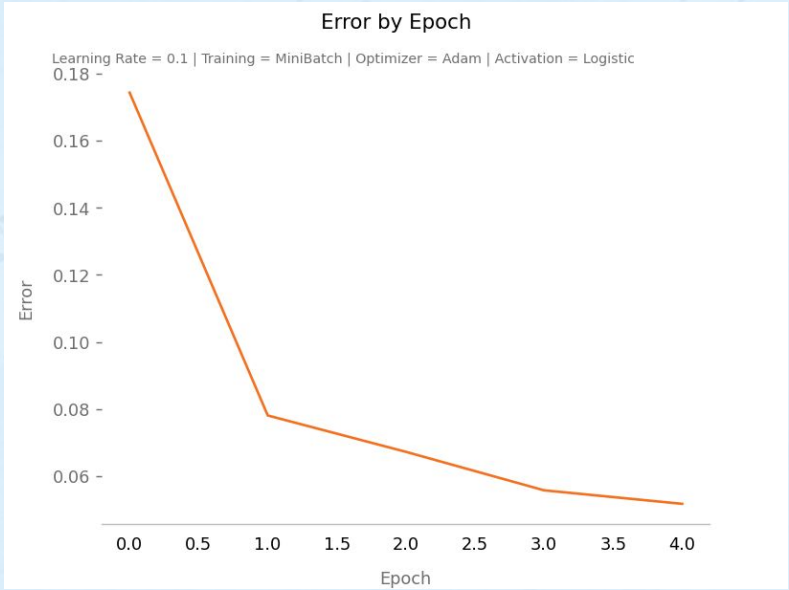
MNIST

Error

Primer Intento



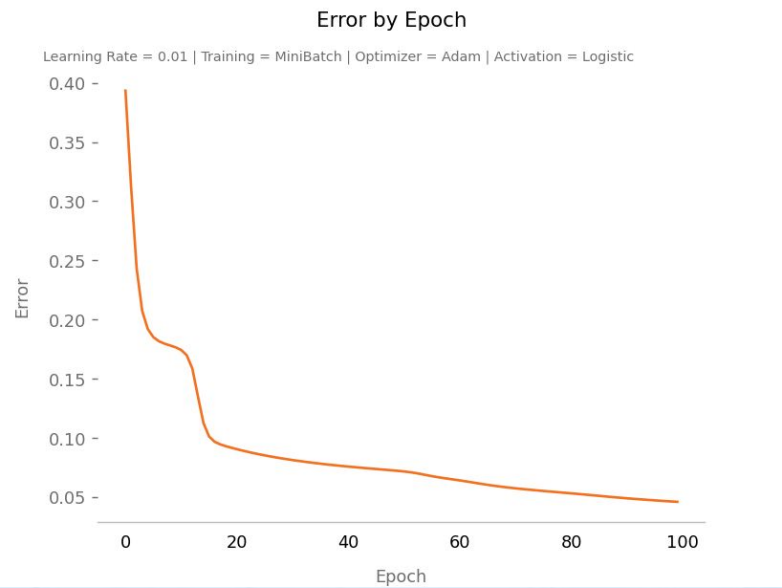
Tercer Intento



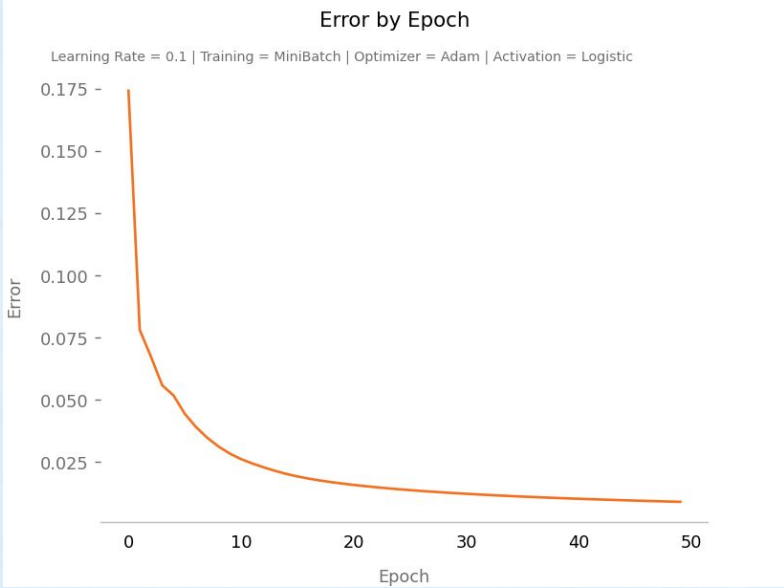
MNIST

Error

Segundo Intento



Cuarto Intento



CONCLUSIONES

- La arquitectura es el factor **diferencial** cuando se trata de evaluar la capacidad de generalización del conjunto de datos de *MNIST*

- Son conjuntos de datos muy grandes, se nota mucho que a medida que incrementa el tamaño del input, la red escala muy rápidamente en una estructura ineficiente desde el punto de vista del costo computacional.

- La capacidad de generalización la determina la arquitectura. Pero sin suficiente entrenamiento, el error no se reduce significativamente como para que permita clasificar mejor los datos.

- Hay mucho juego que se puede hacer en la red para aumentar la eficiencia en relación a cómo se va entrenando.
  Ahí entran las librerías como *Pytorch* y su influencia en ayudar a “eficientizar” las arquitecturas y sus operaciones internas.

GRACIAS