Deep Learning

Grupo 4 - Generative autoencoder, DAE & VAE

Autoencoder Clásico

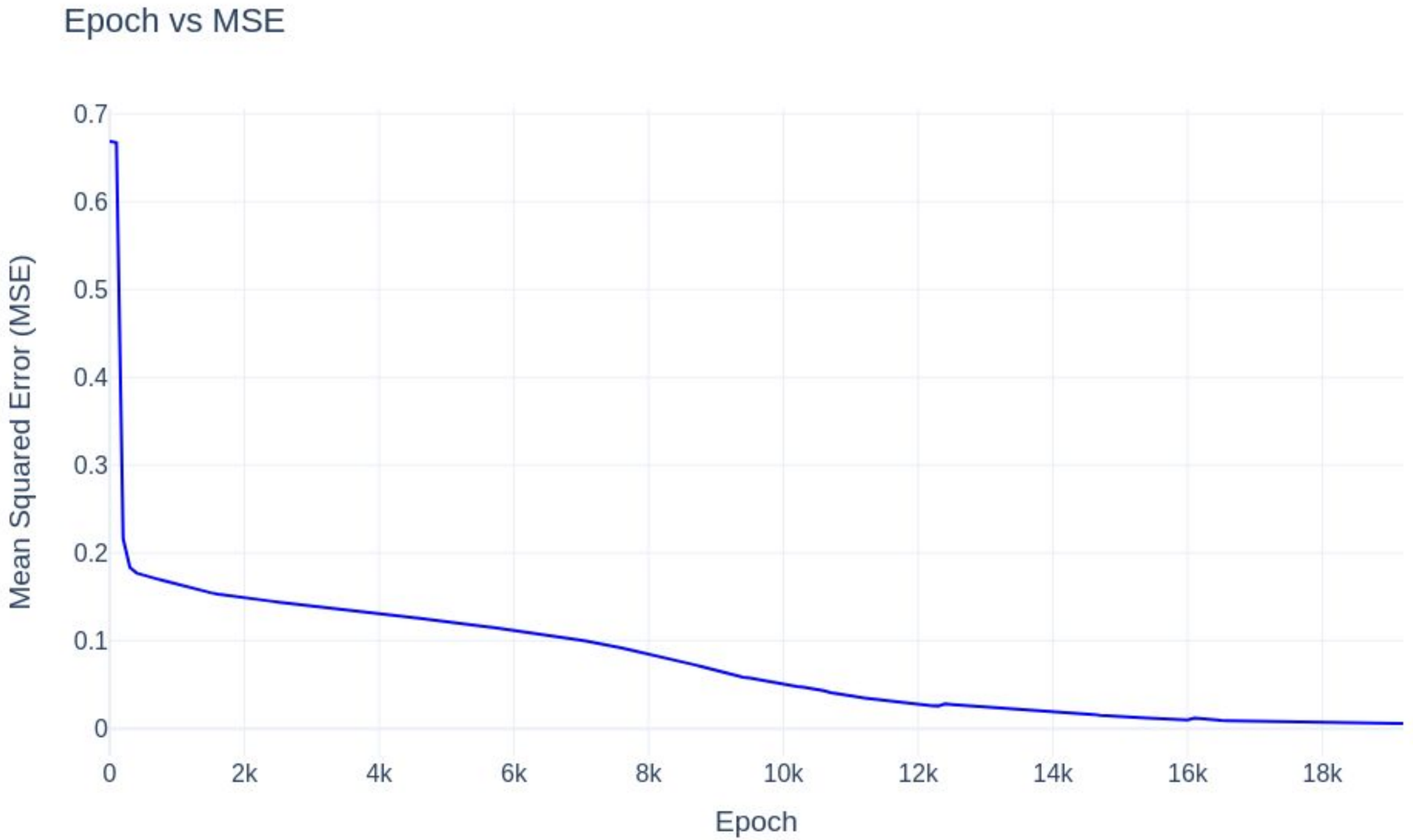
Elección de arquitectura

Para decidir cuál era la mejor arquitectura observamos el Mean Square Error (MSE) en el perceptrón multicapa

Notamos que es más sencillo dar con arquitecturas que converjan a un MSE bajo durante el entrenamiento cuando cuentan con no más de 2 capas ocultas tanto en el encoder como en el decoder

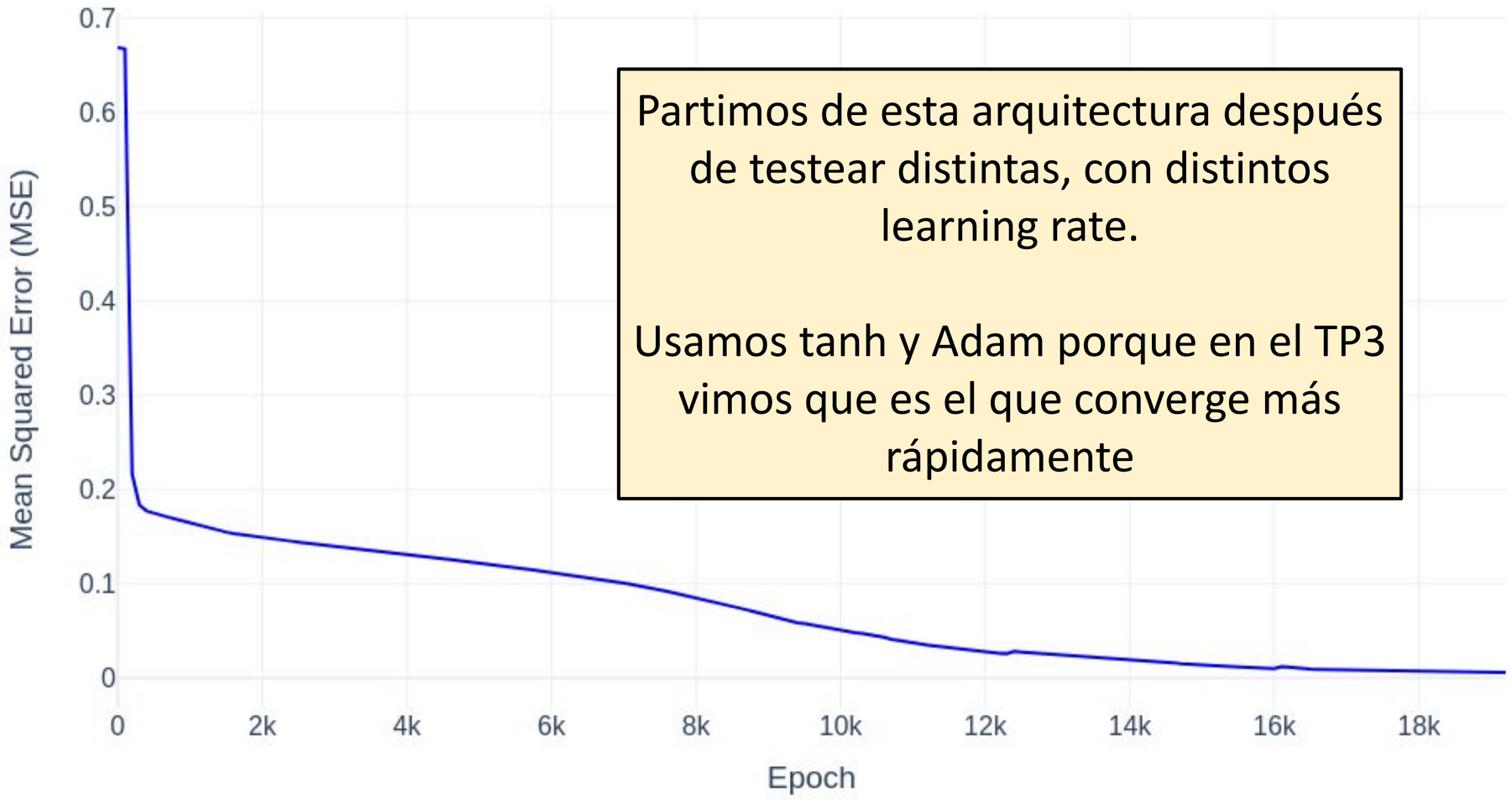
$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Mejor configuración



capas ocultas = [20]
n = 0.0005
epochs = 19300
activation = tanh
optimizer = Adam

Mejor configuración



Partimos de esta arquitectura después de testear distintas, con distintos learning rate.

Usamos tanh y Adam porque en el TP3 vimos que es el que converge más rápidamente

capas ocultas = [20]
n = 0.0005
epochs = 19300
activation = tanh
optimizer = Adam

Interrogantes

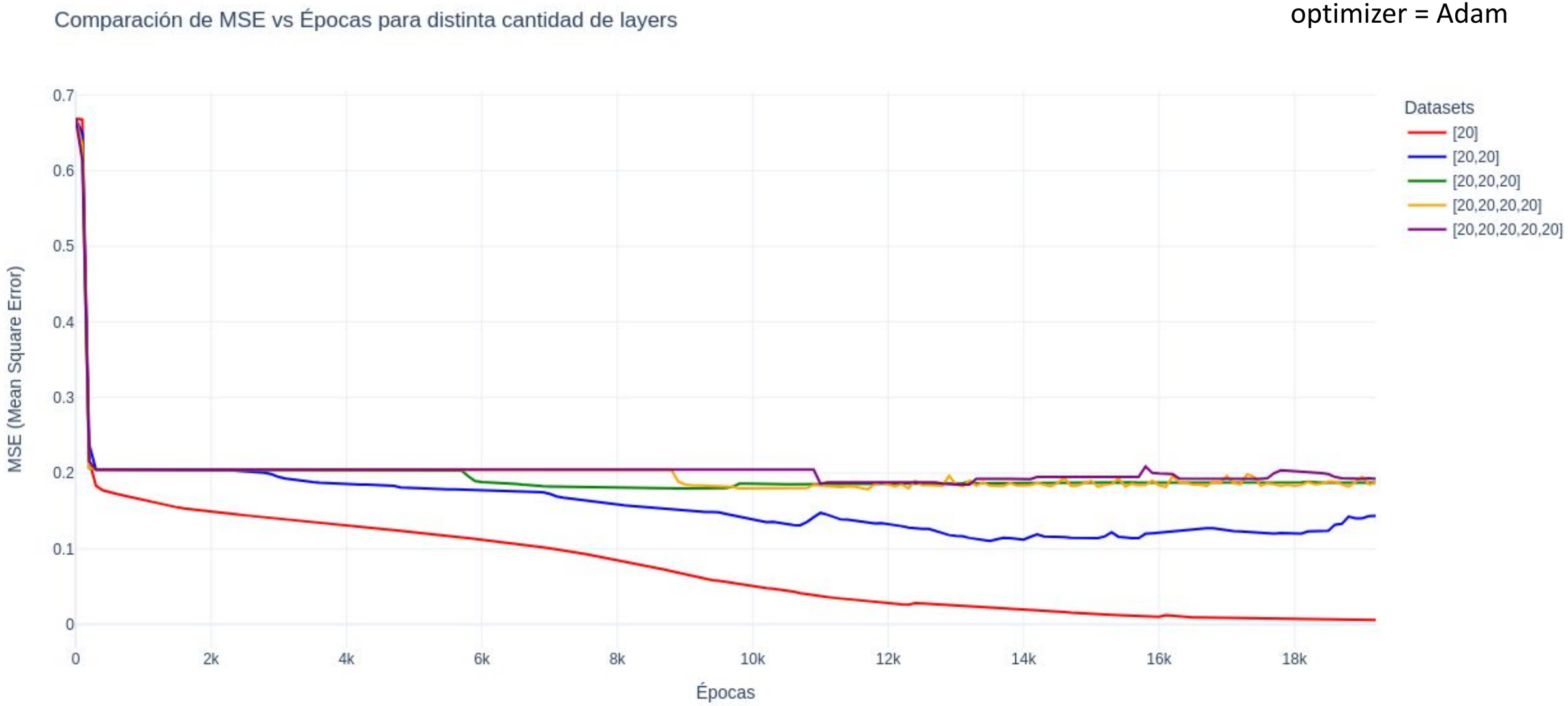
Luego de encontrar la mejor arquitectura que pudimos nos surgieron las siguientes preguntas:

- ¿Realmente son mejores las arquitecturas con menos capas en comparación con las que tienen más?

- Dado que el mse es más chico con un learning rate más chico ¿Será mejor la optimización con Momentum dado que genera una convergencia más lenta que Adam?

Comparación de architectura por cantidad de layers

capas ocultas = [20]
n = 0.0005
epochs = 19300
activation = tanh
optimizer = Adam



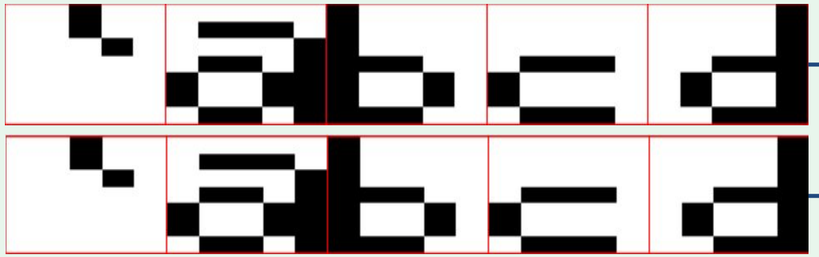
Comparación de optimizers para la mejor configuración

capas ocultas = [20]
n = 0.0005
epochs = 19300
activation = tanh

Resultados y observaciones

- Es complicado encontrar una arquitectura
- Por lo experimentado, la convergencia del error no es correlativa a los parámetros de la arquitectura: Comparado a los trabajos anteriores, en este no se pueden hacer pruebas donde vamos incrementando de a poco los parámetros hasta alcanzar la solución. Existen soluciones “aisladas”. Una determinada arquitectura puede no tener una convergencia del mse pero al aumentar en 1 sola neurona una de las capas podemos tener una convergencia a un mse chico.

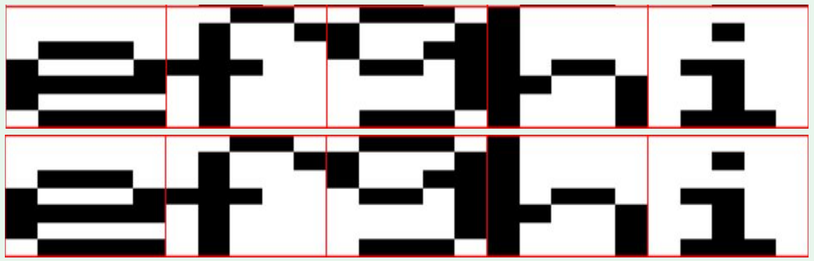
Predicciones



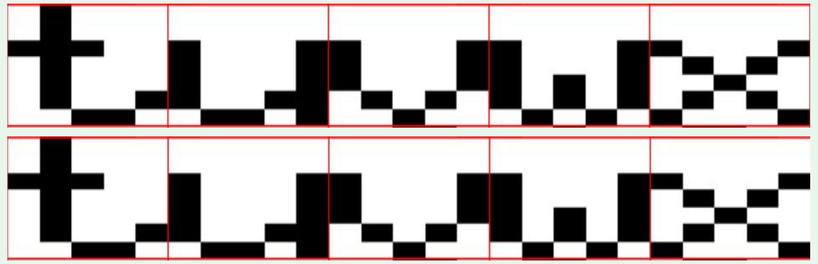
original

predicción

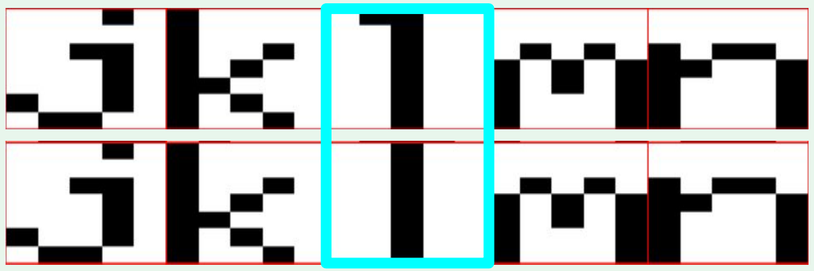
- capas ocultas = [20]
- n = 0.0005
- epochs = 19300
- activation = tanh
- optimizer = Adam

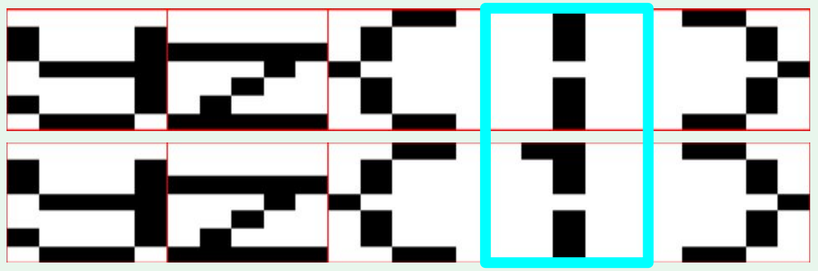


1 Error

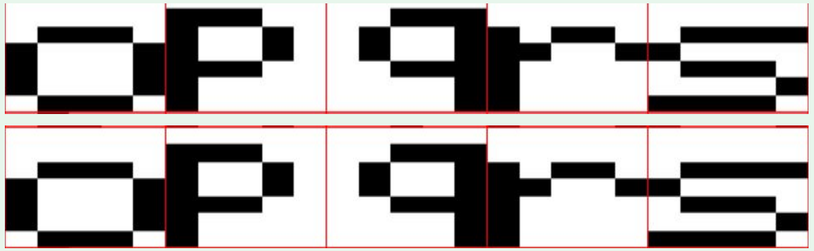


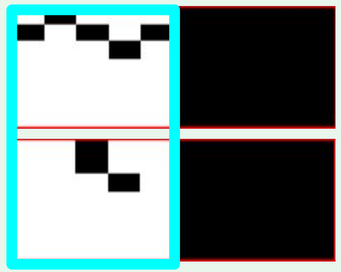
1 Error





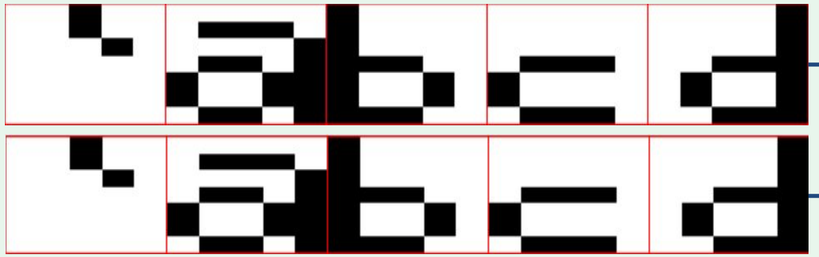
4 Errores: No aprendió el carácter





Lo asoció al acento grave

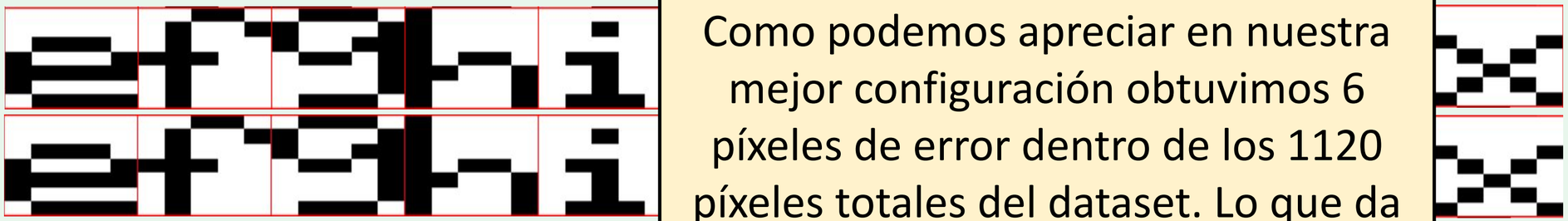
Predicciones



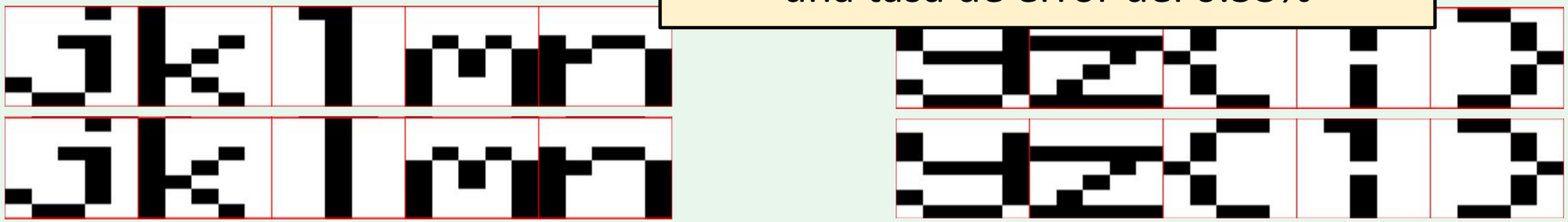
original

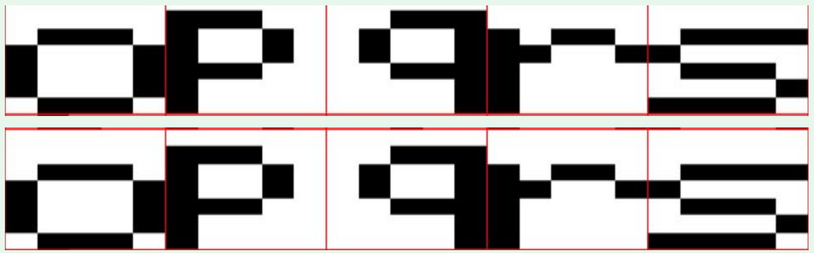
predicción

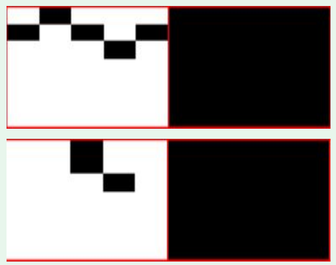
- capas ocultas = [20]
- n = 0.0005
- epochs = 19300
- activation = tanh
- optimizer = Adam



Como podemos apreciar en nuestra mejor configuración obtuvimos 6 píxeles de error dentro de los 1120 píxeles totales del dataset. Lo que da una tasa de error del 0.53%

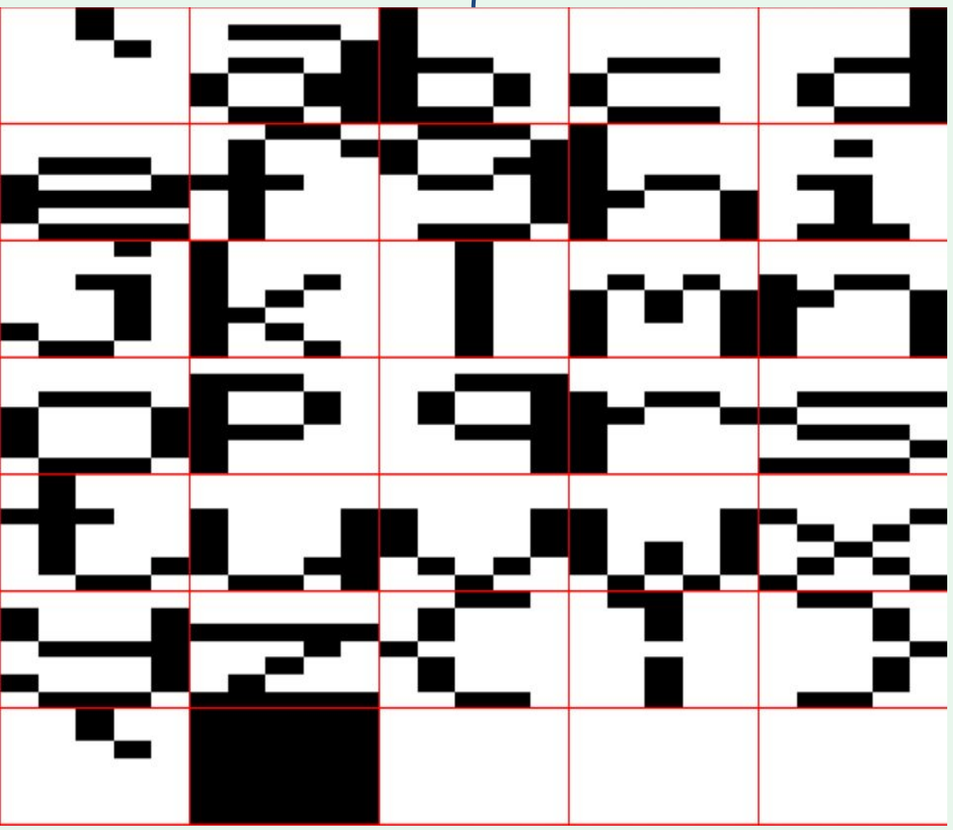


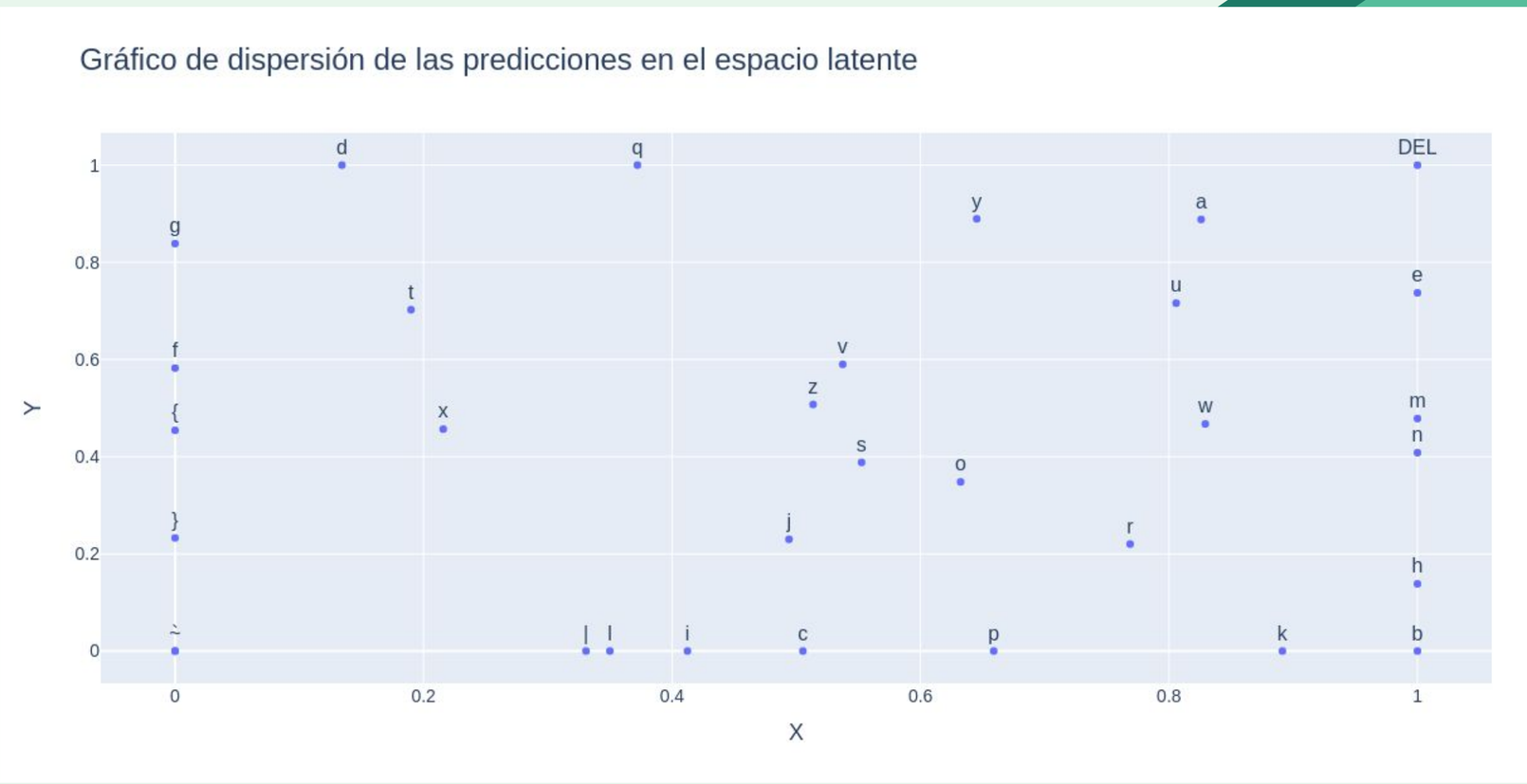




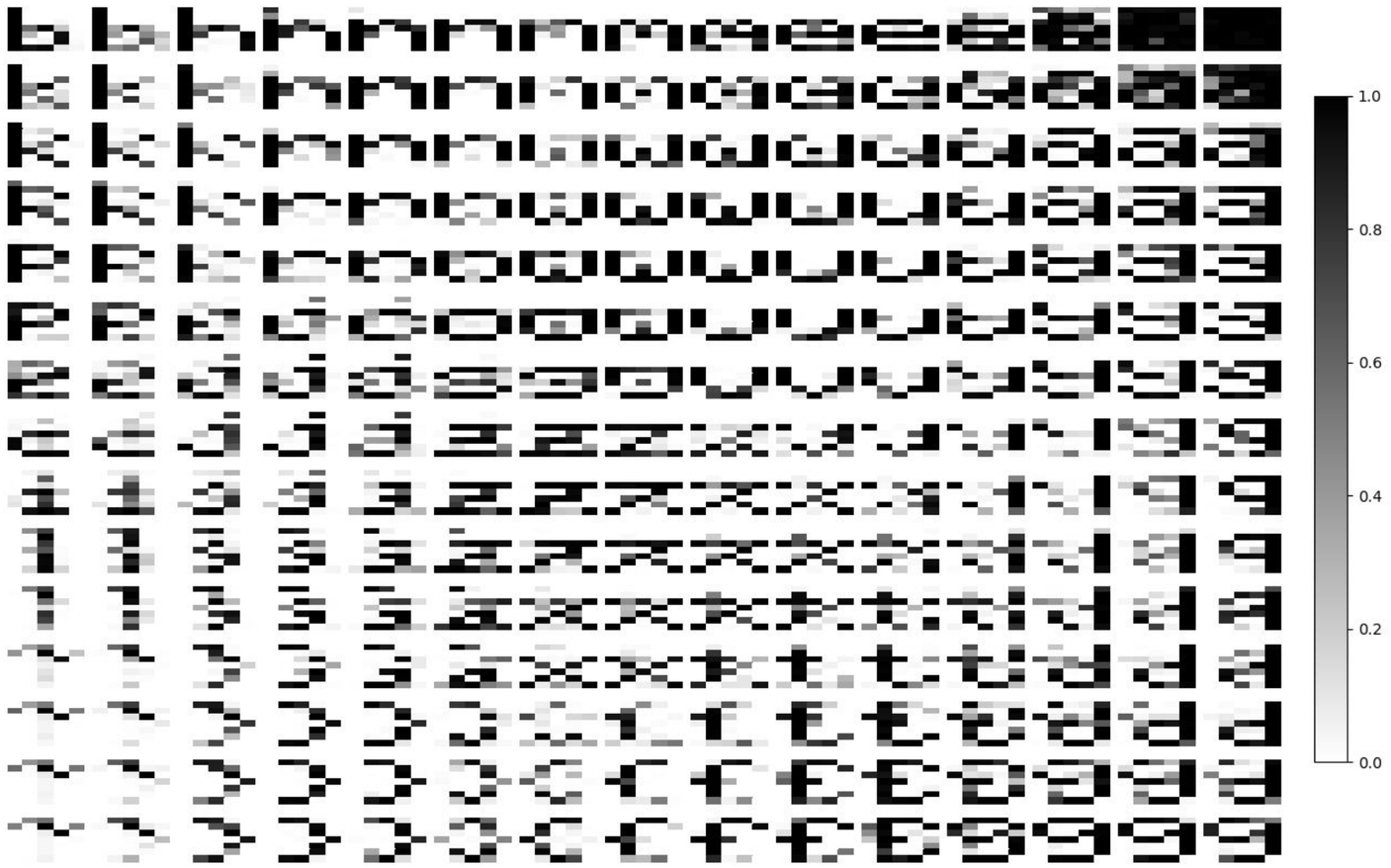
Distribución del dataset en el espacio latente

Predicciones





Generación de nuevas letras con el autoencoder clásico



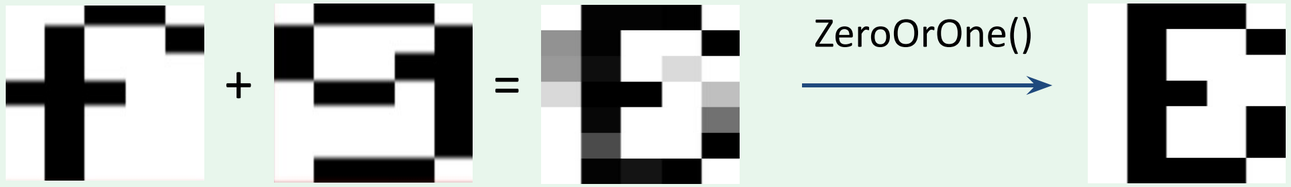
Tomamos un array de 15 puntos equidistantes entre 0 y 1 y formamos una mapa de coordenadas de 15x15=225 puntos.

Reconstruimos dichos puntos para encontrar los pares de caracteres que generan los nuevos más interesantes.

Las combinaciones más interesantes son: (f,g), (o,w)

Caracteres encontrados

Se tomaron dos caracteres cercanos en el espacio latente y se trazó una línea por la cual nos fuimos moviendo para elegir distintos puntos que fuimos representando utilizando el decoder del autoencoder entrando. Luego “limpiamos” las representaciones con la función ZeroOrOne.



“e**feg**e”

La letra “feg” es más bien parecida a una ‘E’ con la diferencia de que no es simétrica

“odo**ble**v”

La letra “ble” es como un caracter omega con la parte de arriba “cerrada”

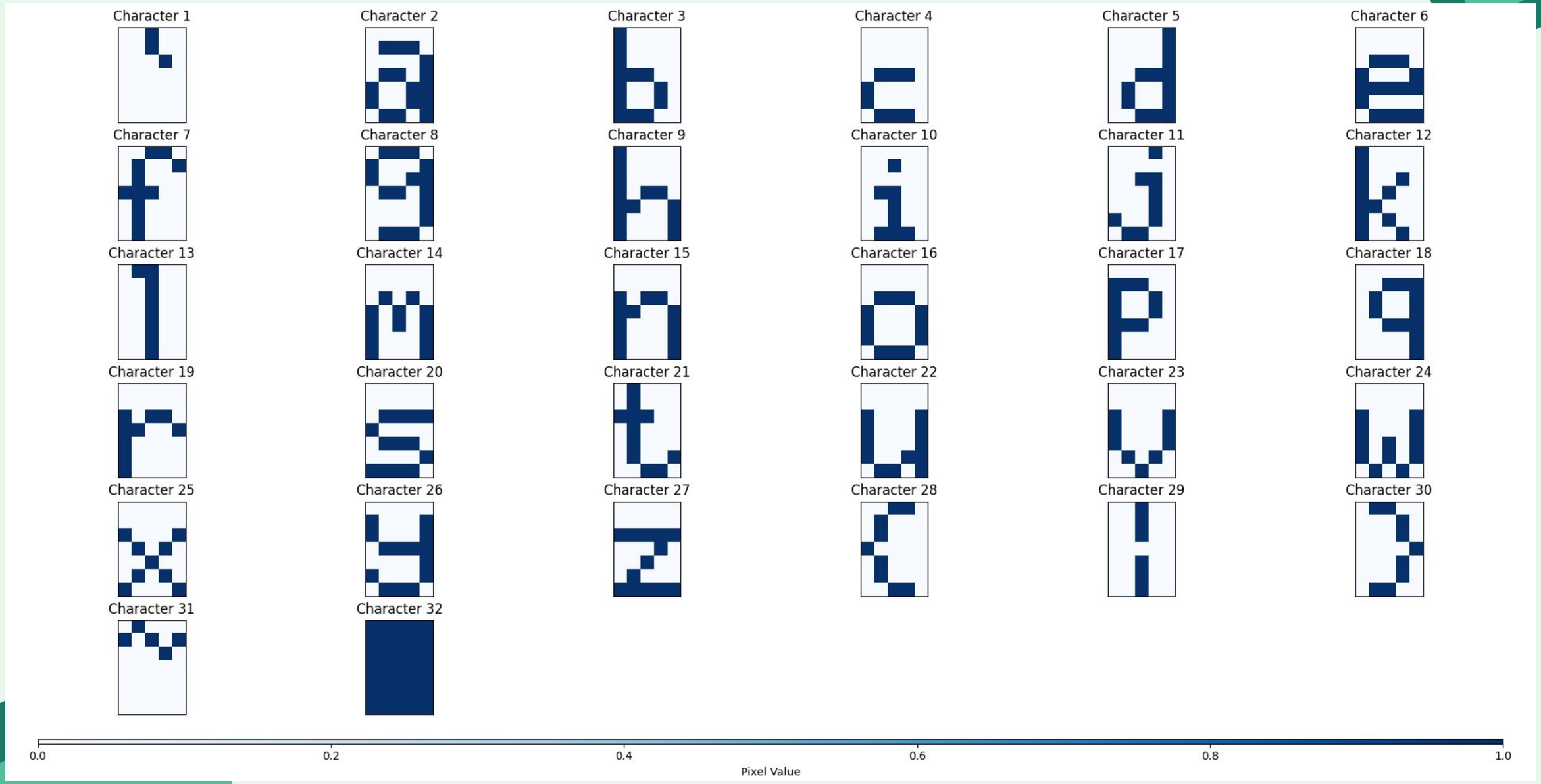
Denoising Autoencoder

Preparación

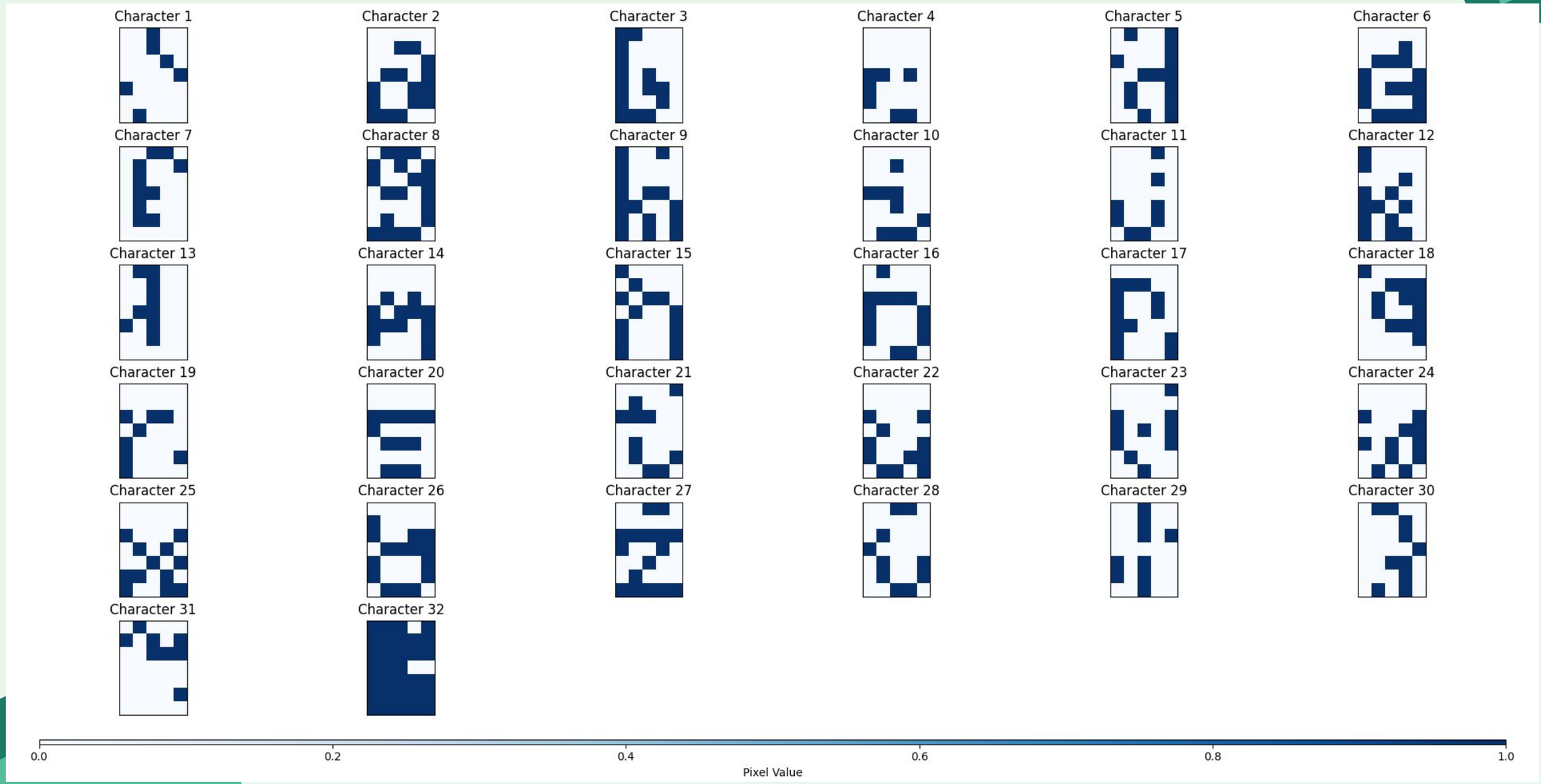
- Se optó por usar Salt and Pepper (con 0.1 de noise_level) como ruido para la matriz de caracteres.

- Se aplicó dicho ruido para cada uno de los caracteres.

Caracteres sin ruido



Caracteres con ruido



Idea del Denoising

Entrada con ruido

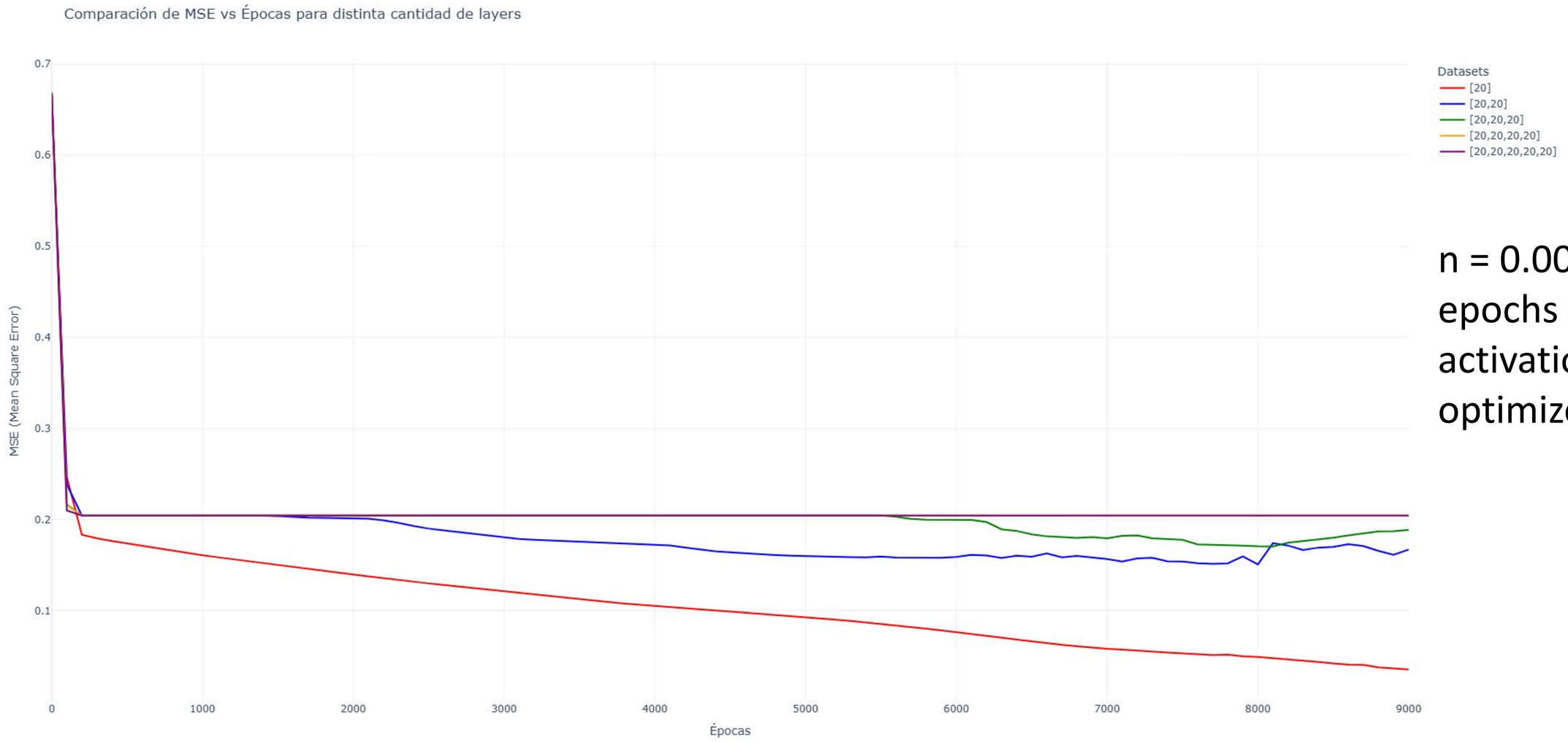
Salida esperada

Elección de arquitectura

- Similar a lo que se realizó con el autoencoder normal.
- Se comparó el MSE (Mean Squared Error) en el perceptrón multicapa.
- Se notó que con un número adecuado de capas ocultas se obtiene la mejor performance.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Mse vs Épocas

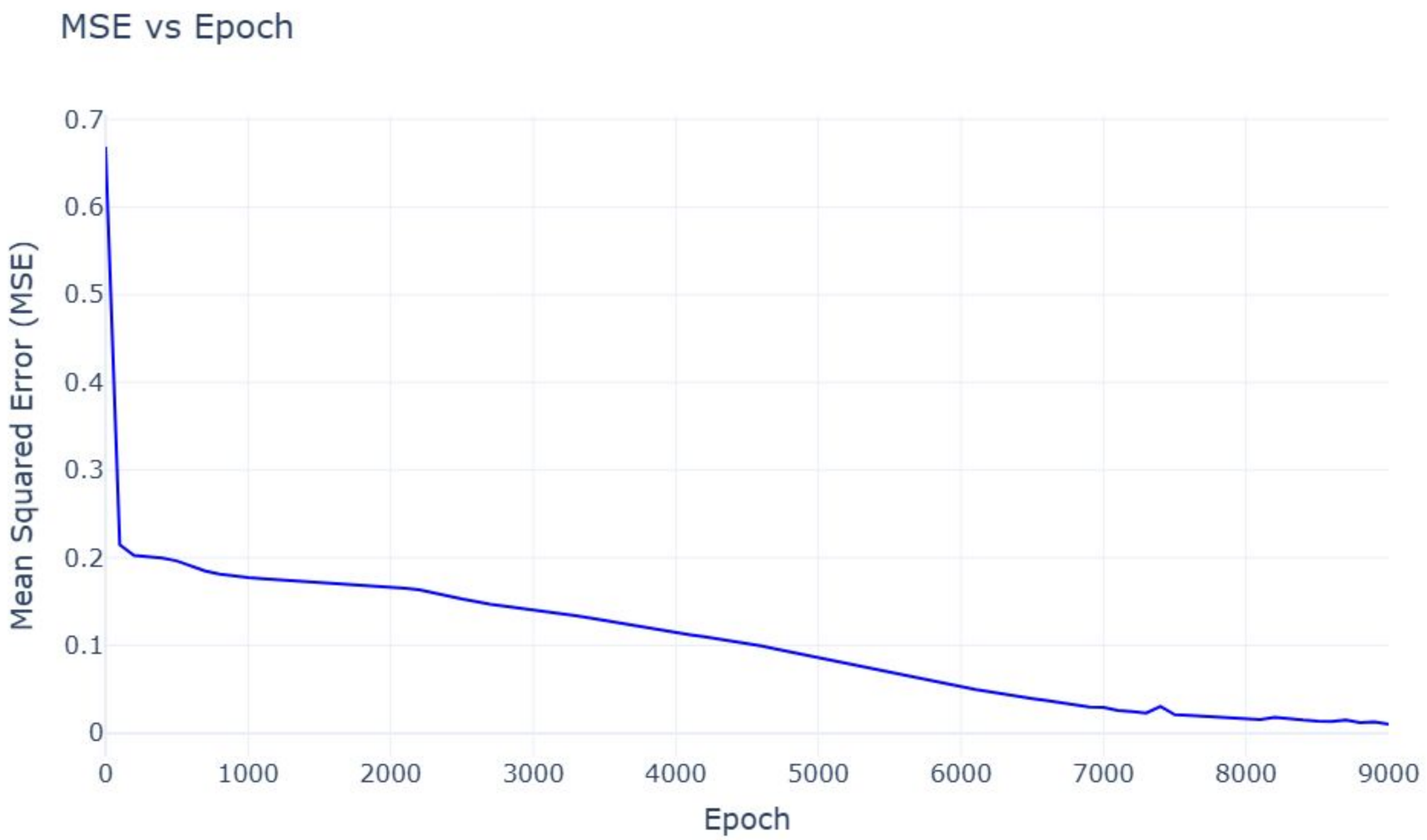


n = 0.001
epochs = 9100
activation = tanh
optimizer = Adam

Comparación de optimizadores

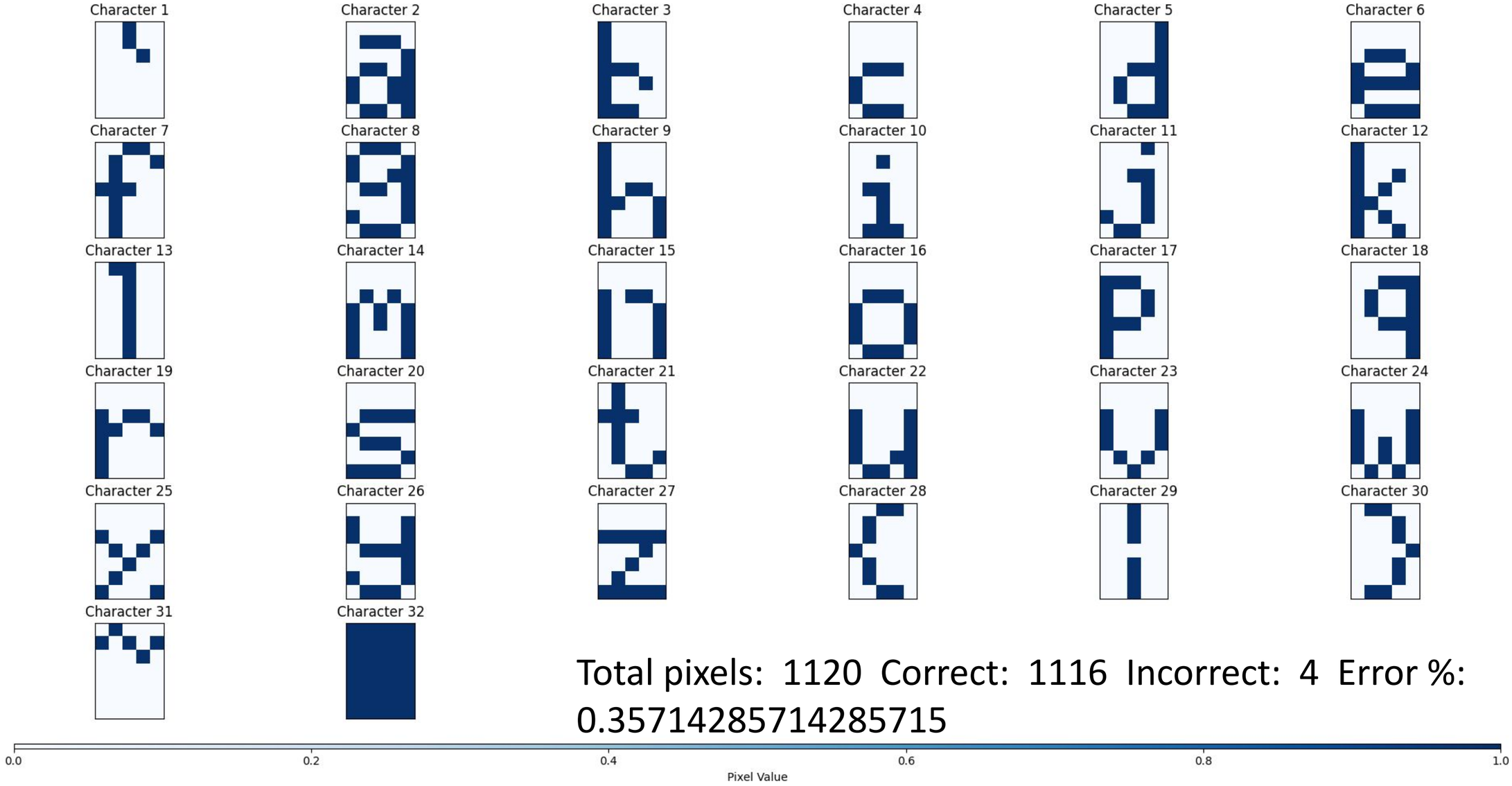
Comparación de learning rates

Mejor configuración



arquitectura = [35,32,8,2,8,32,35]
capas ocultas = [32,8]
n = 0.001
epochs = 9100
activation = tanh
optimizer = Adam

Predicción del autoencoder



Total pixels: 1120 Correct: 1116 Incorrect: 4 Error %: 0.3571428571428571

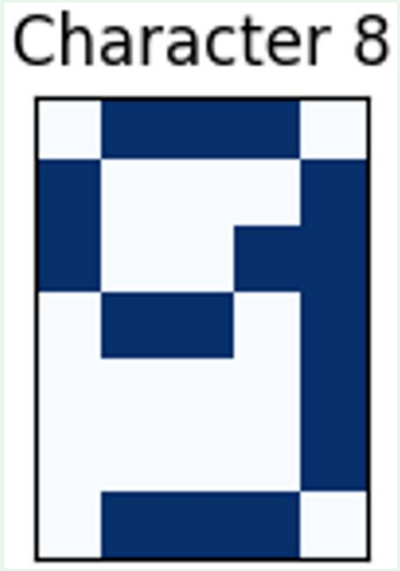
Predicción del autoencoder

Original

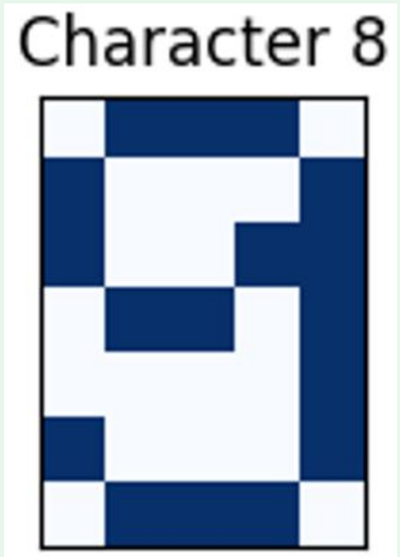
Predicción

Predicción del autoencoder

Original

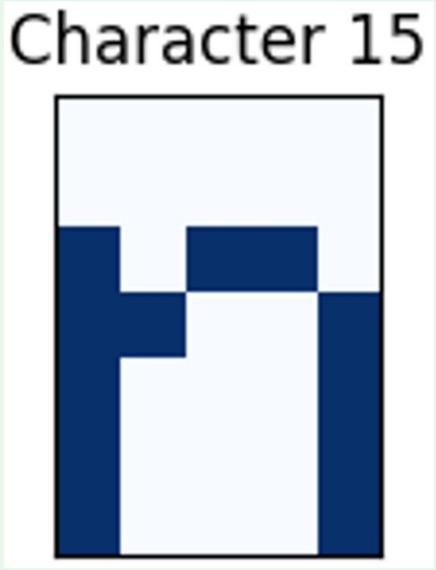


Predicción

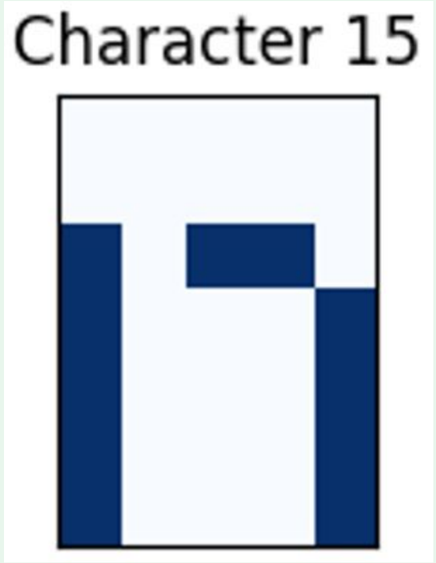


Predicción del autoencoder

Original

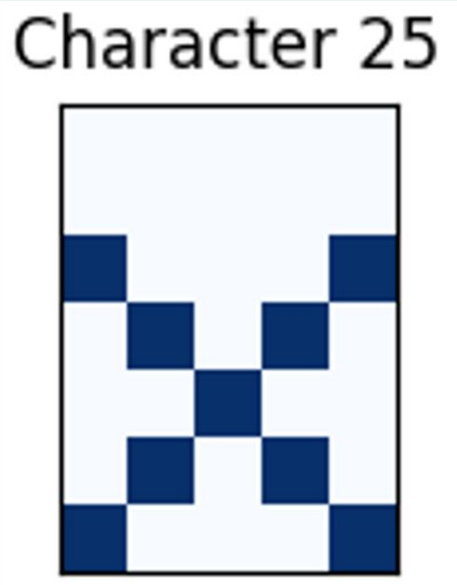


Predicción

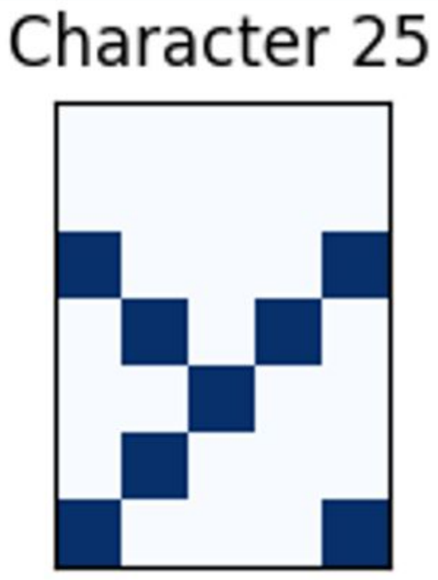


Predicción del autoencoder

Original



Predicción



Conclusiones

- Viendo los resultados se puede concluir que realiza una gran aproximación ya que solo 4 letras tuvieron un píxel de error.
- Al igual que en el ítem anterior, no se puede definir la mejor arquitectura viendo solo el MSE.
- Incluso con caracteres que son similares aplicándoles ruido (como pueden ser la h y la n) los distingue correctamente.

Autoencoder Variacional

Dataset 1

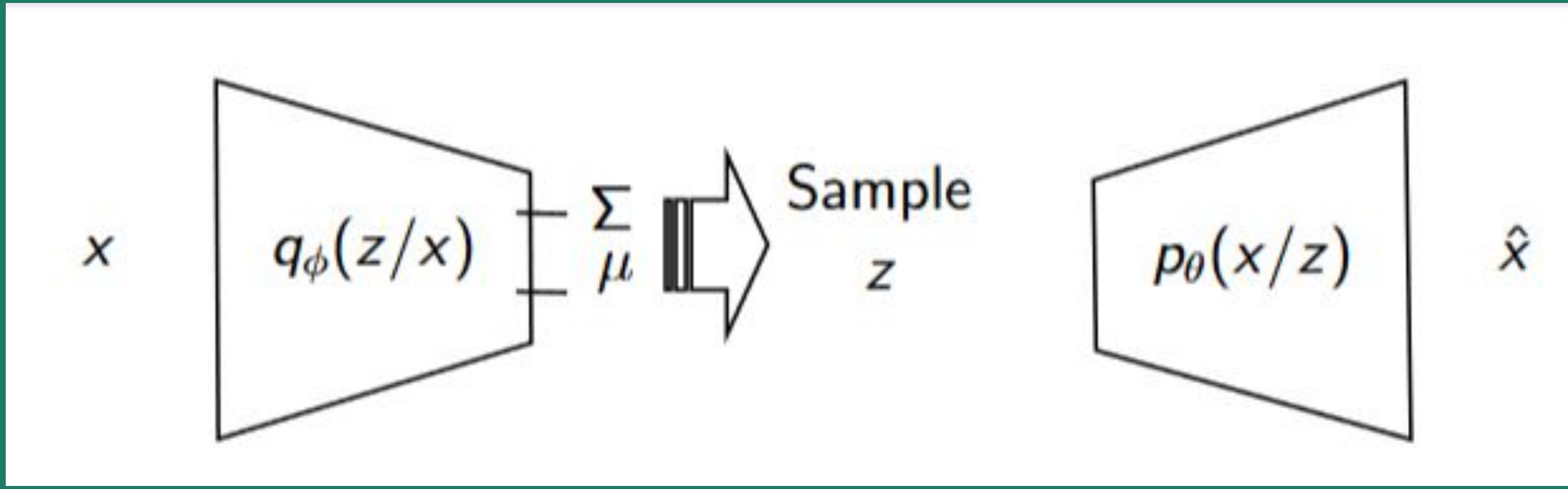
8 Emojis:

Dataset 1

20x20 pixels c/u (grayscale)

Se aplanan los 400 píxeles en un vector unidimensional.

Arquitectura



Para elegir la arquitectura se comparó el total loss.
Se busco la arquitectura que **MINIMICE**:

$$-\mathcal{L} = -\underbrace{\mathbb{E}_{q(z)} \log p(x/z)}_{\text{Error de reconstrucción}} + \underbrace{KL(q(z)||p(z))}_{\text{Término regularizador}}$$

Hiperparametros

Se testean las siguientes arquitecturas:
[400,200,**2**,200,400]
[400,200,100,**2**,100,200,400]
[400,50,30,**2**,30,50,400]
[400,150,50,15,**2**,15,50,150,400]

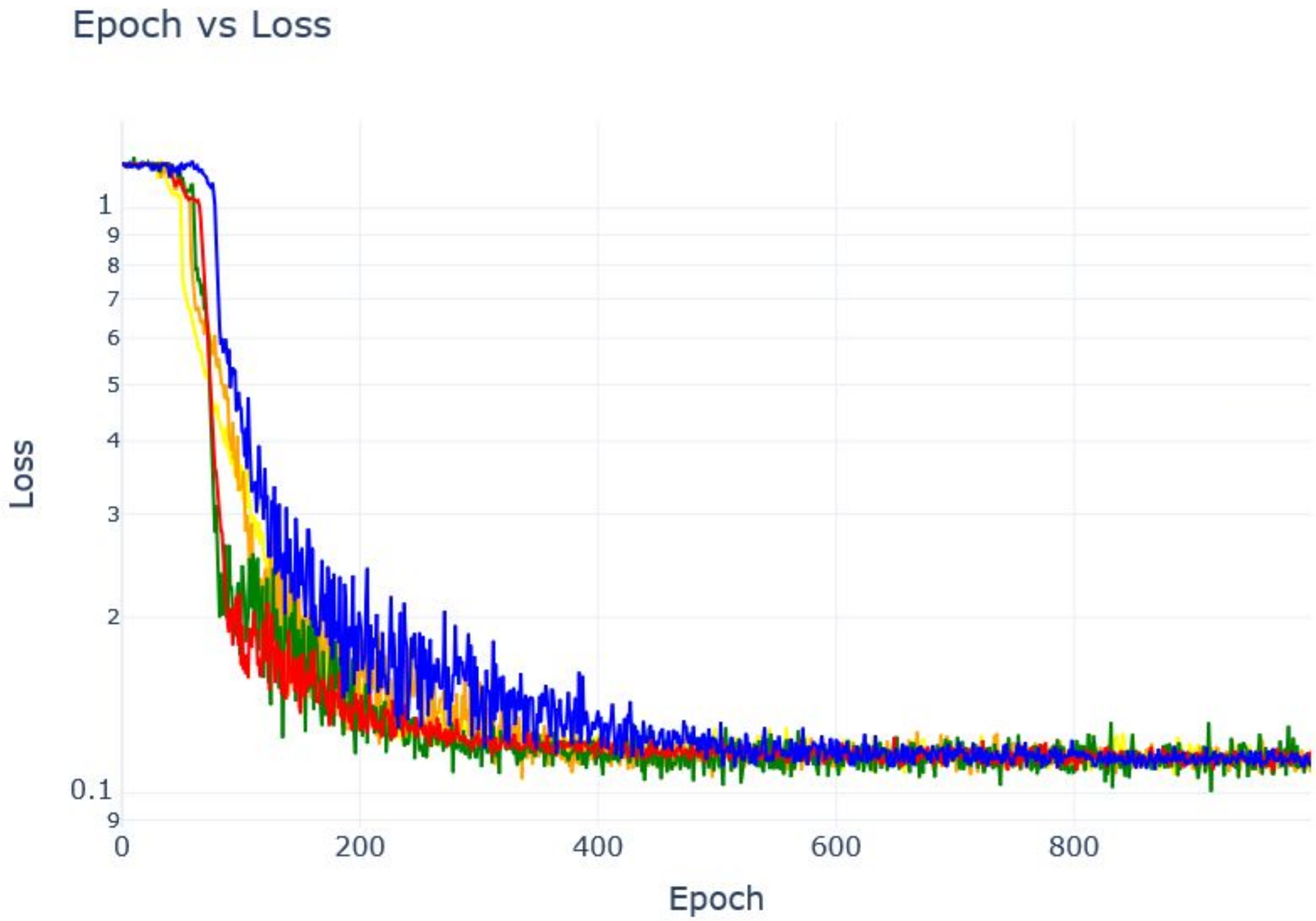
Se usara ADAM con b1=0.9 y b2=0.999. Se testeara su learning rate:
[0.1,0.01,0.001,0.0001]

Full Batch

Funcion de activacion: TanH

Epocas: 1000

Loss para distintas arquitecturas



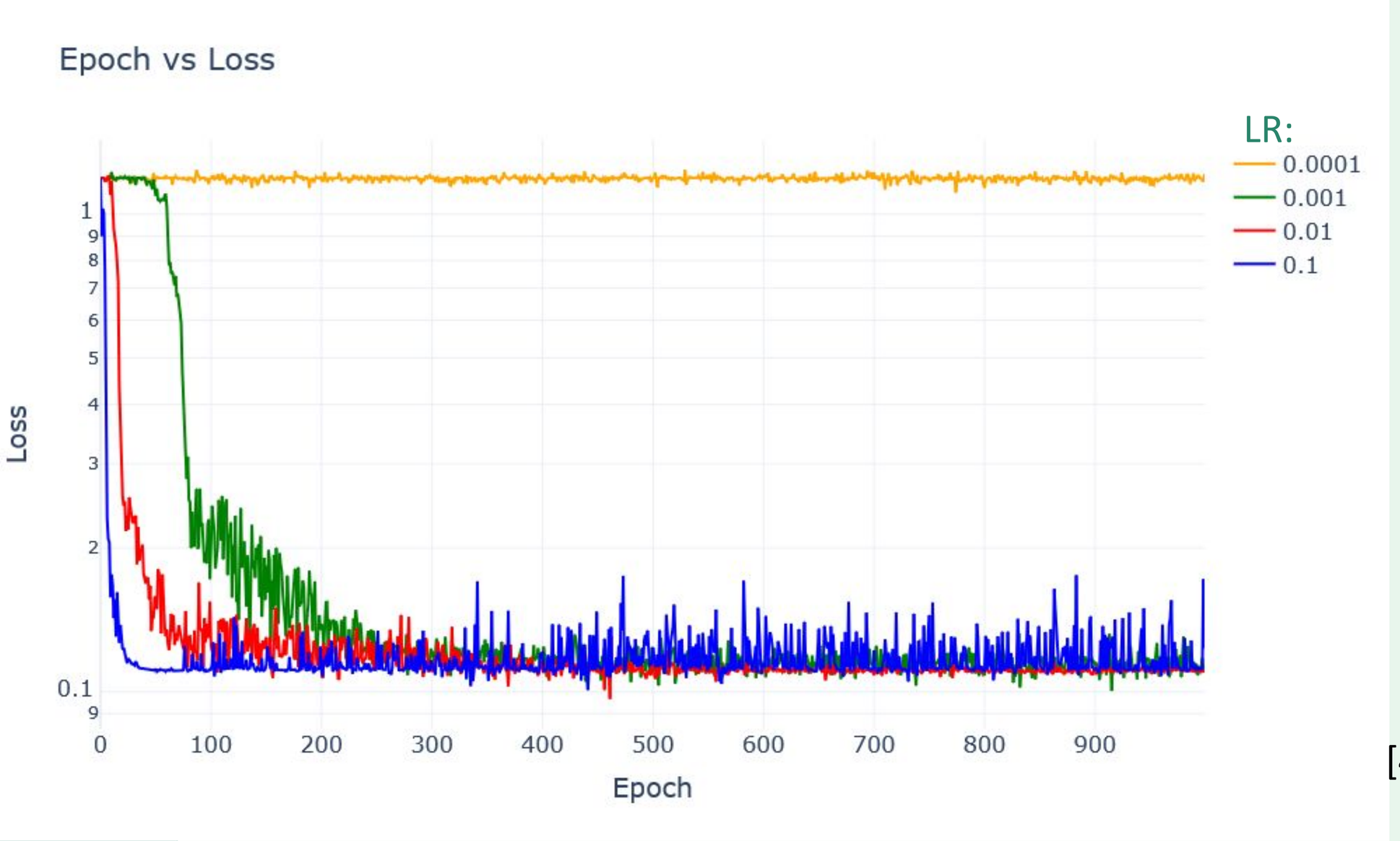
Neuronas por capa del encoder

- 400,400,200,100
- 400,200,100
- 400,200
- 400,150,50,15
- 400,50,30

En el decoder son las mismas neur
por capa pero invertido

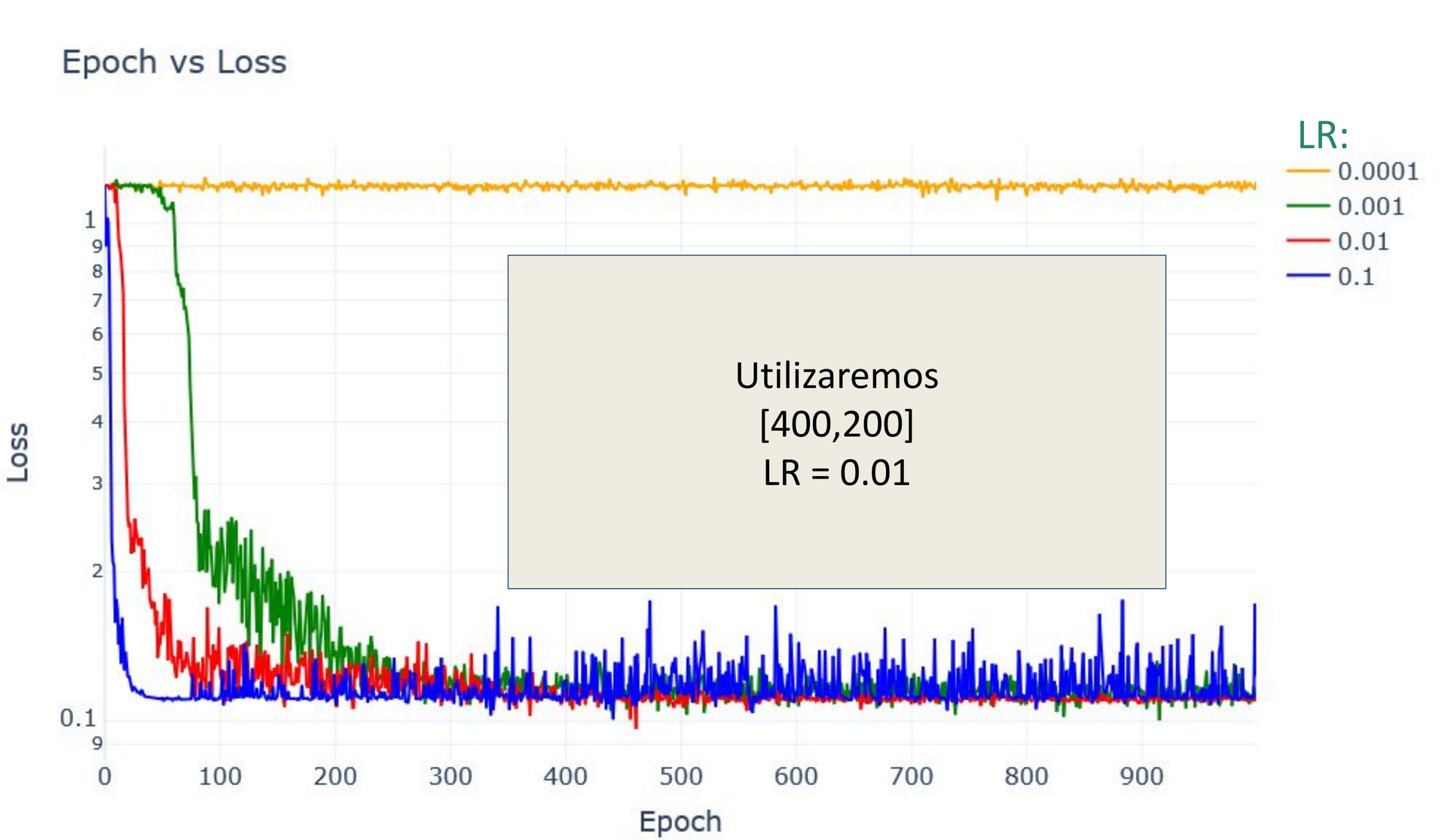
Funcion: TanH
Epocas: 1000
Learning Rate = 0.001

Loss para distintos LR



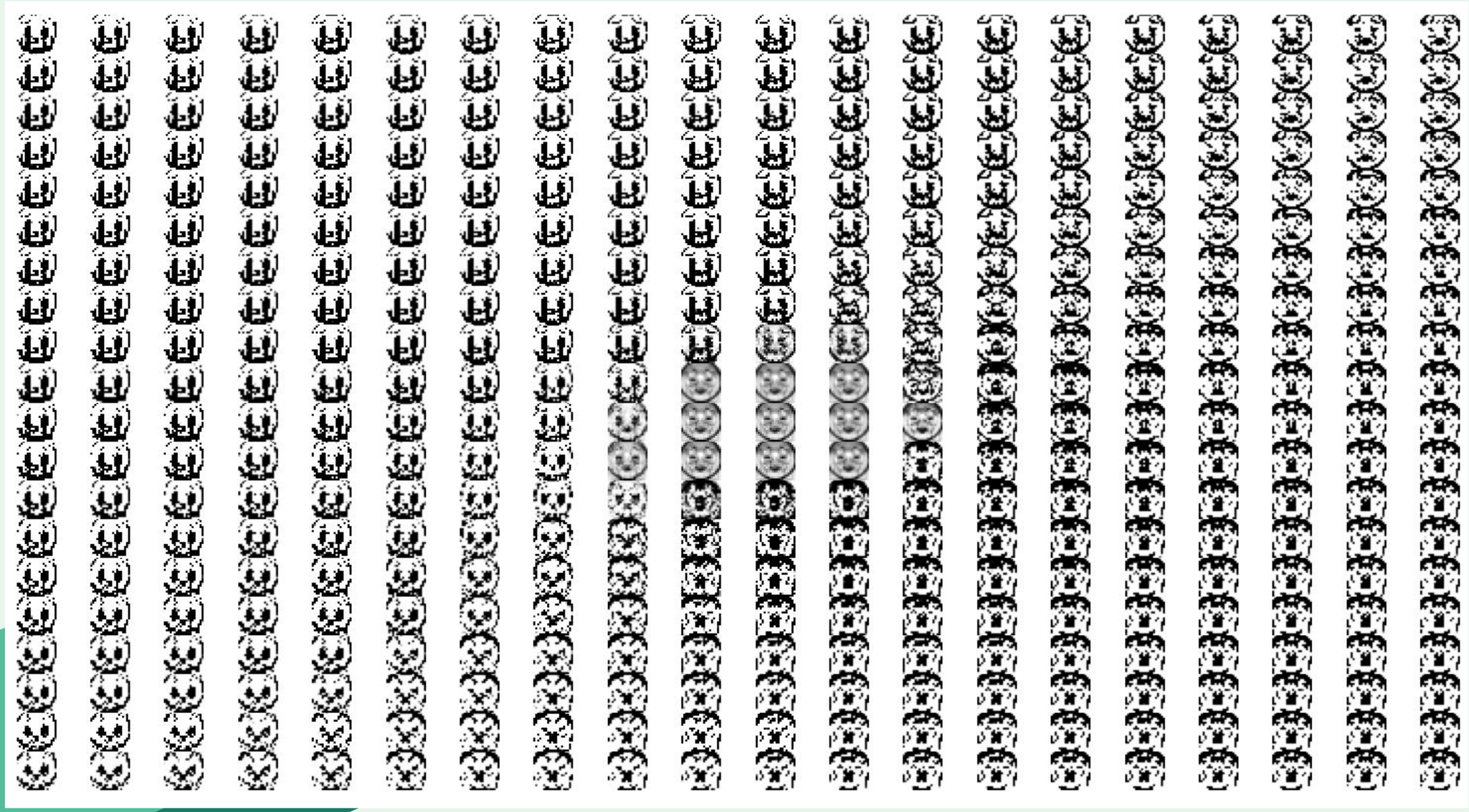
Funcion: TanH
Epocas: 1000
Arquitectura = [400,200,2,200,400]

Loss para distintos LR



Funcion: TanH
Epocas: 1000
Arquitectura = [400,200,2,200,400]

Emojis generados



Funcion: TanH
Epocas: 1000
Arquitectura = [400,200,2,200,400]
Learning Rate = 0.01

Emojis interesantes




Dataset 2

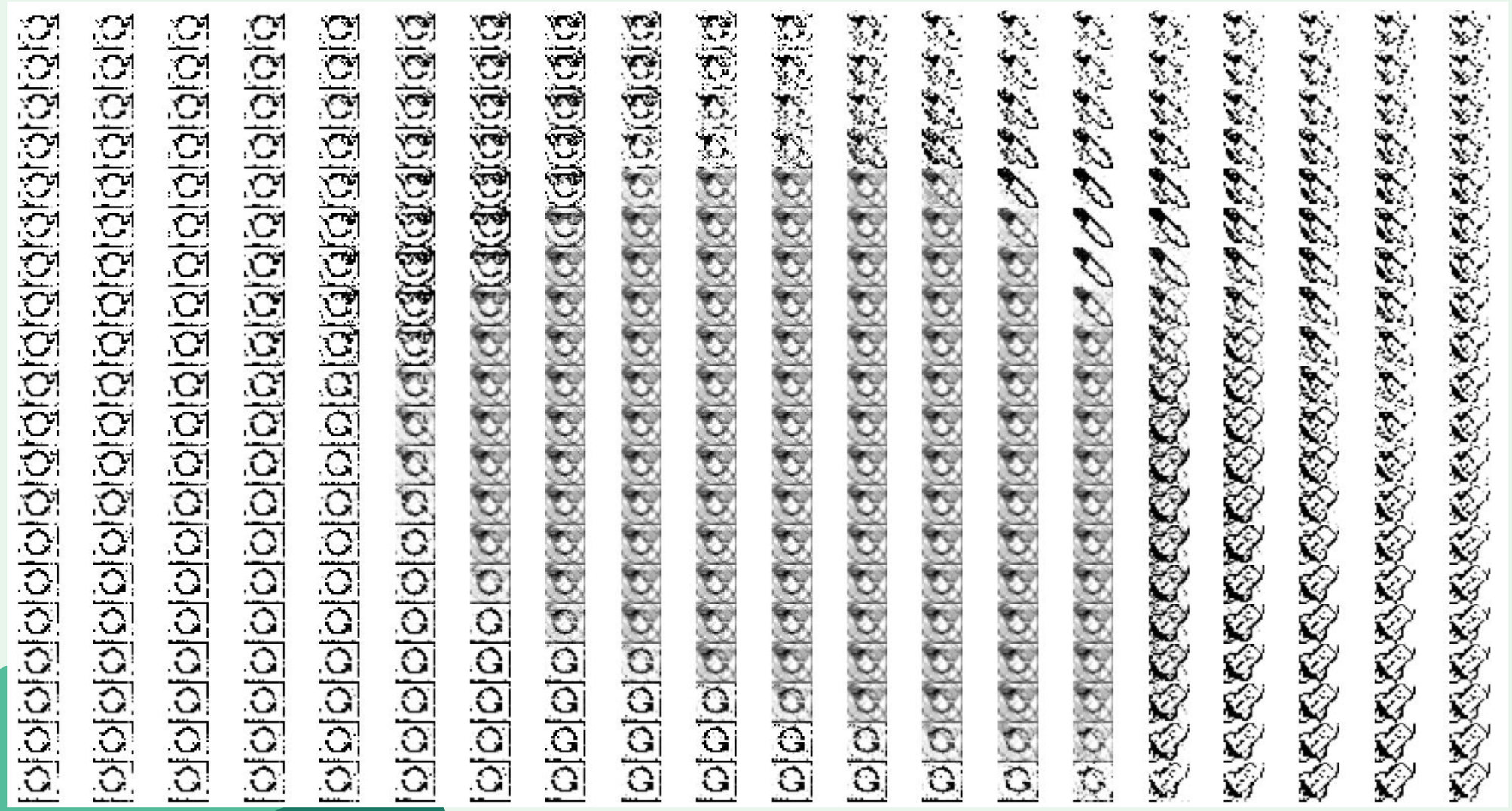
4 Emojis, muy diferentes entre si:

Dataset 2

20x20 pixels c/u (grayscale)

Se aplanan los 400 píxeles en un vector unidimensional.

Emojis generados



Funcion: TanH
Epocas: 1000
Arquitectura = [400,200,2,200,400]
Learning Rate = 0.01

Emojis interesantes

Muchas gracias!