

# Trabajo Práctico N° 5

## Autoencoders

### Grupo 7:

- ❖ Andres Podgorny
- ❖ Hugo Lichtenberger
- ❖ Ouss Slaitane
- ❖ Nicolás Birsa
- ❖ Valentin Ye Li

# Ej1\_a

## Autoencoders

- ❖ ***Arquitectura***
- ❖ ***Optimización***
- ❖ ***Espacio latente***
- ❖ ***Nuevos símbolos***



# Problema - Implementación

Se busca implementar un **Autoencoder** que pueda reconocer a la salida un conjunto de símbolos de entrada, que **no deben tener más de un pixel erróneo** para ser tomados como correctos.

Para la implementación usamos el MLP desarrollado en anteriores TPs y trabajamos con las versiones “*flattened*” de los símbolos.

# Pruebas de Optimización

Se probaron distintas formas de optimizar el Autoencoder, tanto en **calidad de resultados** como **eficiencia** para obtenerlos.

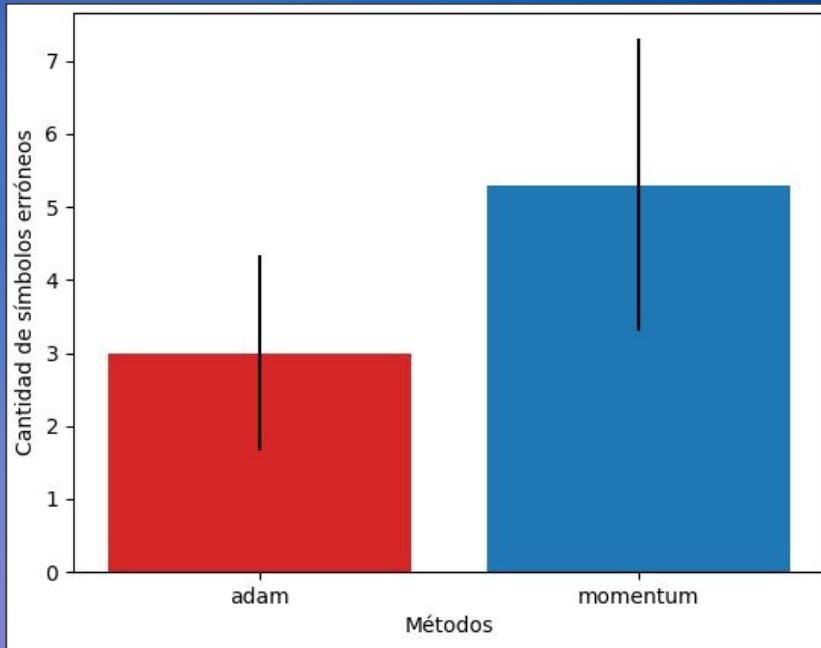
- **Momentum vs Adam**
- **$\eta$  fijo vs  $\eta$  adaptativo**

Cabe destacar que la Arquitectura “idónea” para cada caso es distinta tanto en cantidad de capas como de perceptrones en cada capa, por lo que luego de las pruebas se mostrará la finalmente usada con sus resultados.

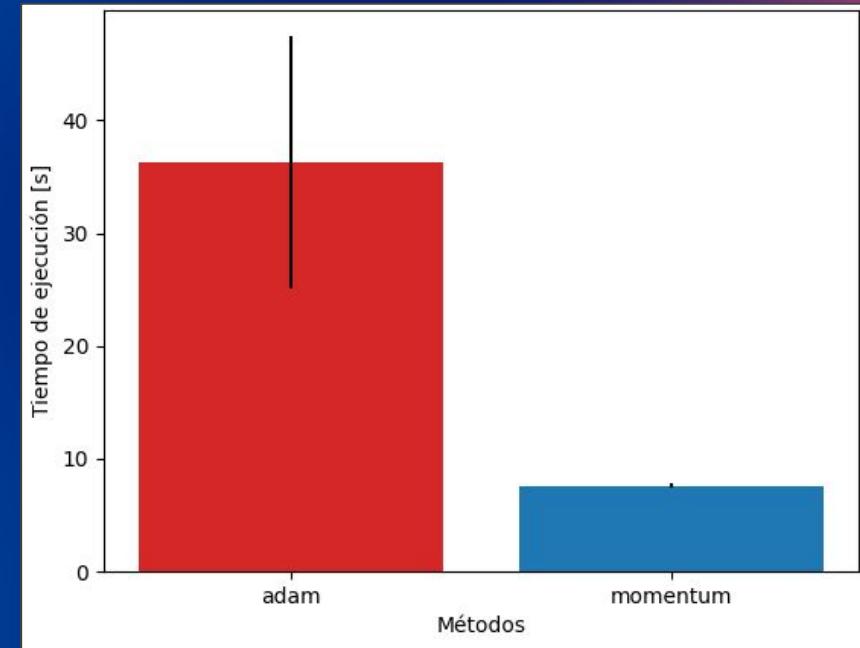
# Momentum vs Adam

10 realizaciones

Símbolos erróneos



Tiempo de ejecución



Momentum: 0.9

Adam:  $\beta_1=0.9$ ,  $\beta_2=0.999$ ,  $\epsilon=10^{-8}$

| 20000 epochs |  $\eta = 0.005$

| 60000 epochs |  $\eta = 0.0005$

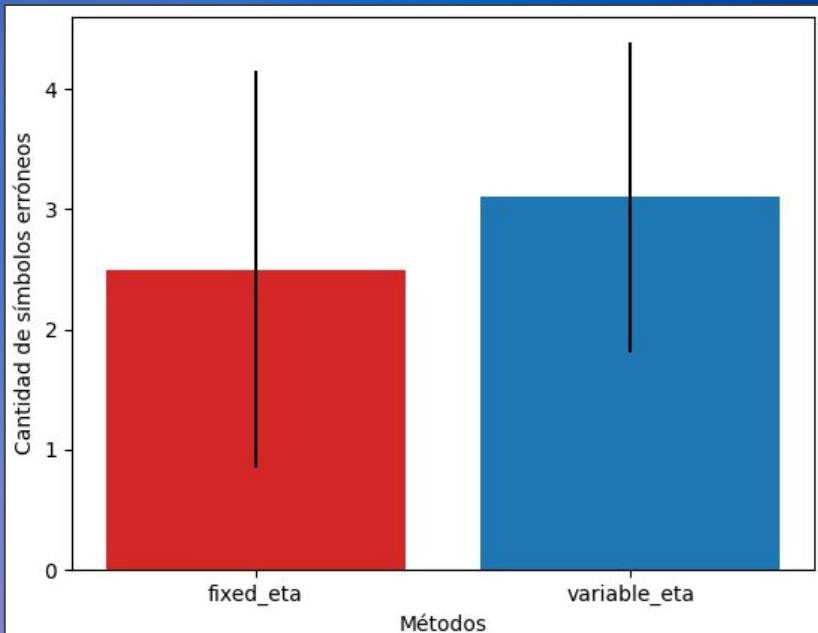
| Capas: [22, 10]

| Capas: [26, 17, 9]

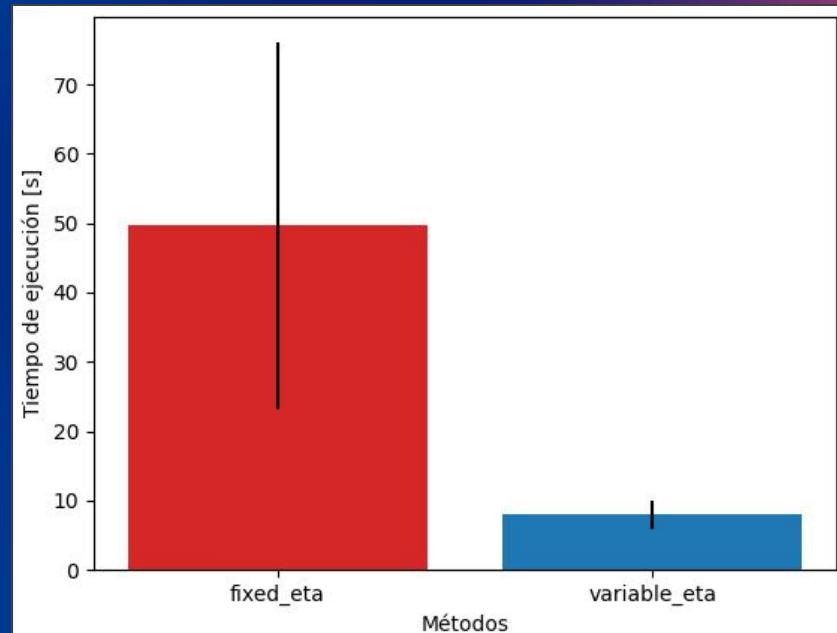
# $\eta$ fijo vs $\eta$ adaptativo

10 realizaciones  
Usando Adam

Símbolos erróneos



Tiempo de ejecución



Fijo: 100000 epochs |  $\eta = 0.0005$   
Adaptativo: 10000 epochs |  $\eta_i = 0.003$  |  $\eta_f = 0.00075$

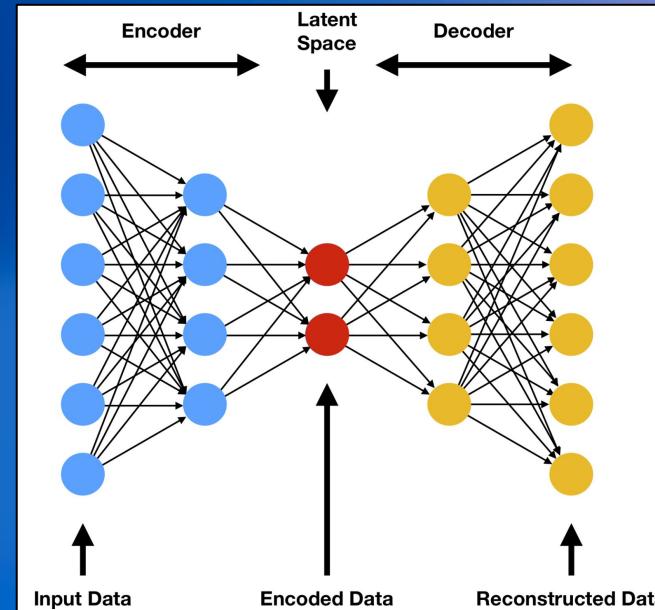
| Capas: [26, 17, 9]  
| Capas: [28, 22, 17, 10]

# Arquitectura

Por lo visto, la arquitectura a usar será de una red con capas:

[35, 28, 22, 17, 10, 2, 10, 17, 22, 28, 35]

Además, se usarán los métodos de optimización de **Adam** y **Eta adaptativo** para todas las futuras pruebas.



## Símbolos erróneos - ejemplo

## Original



## Prediction

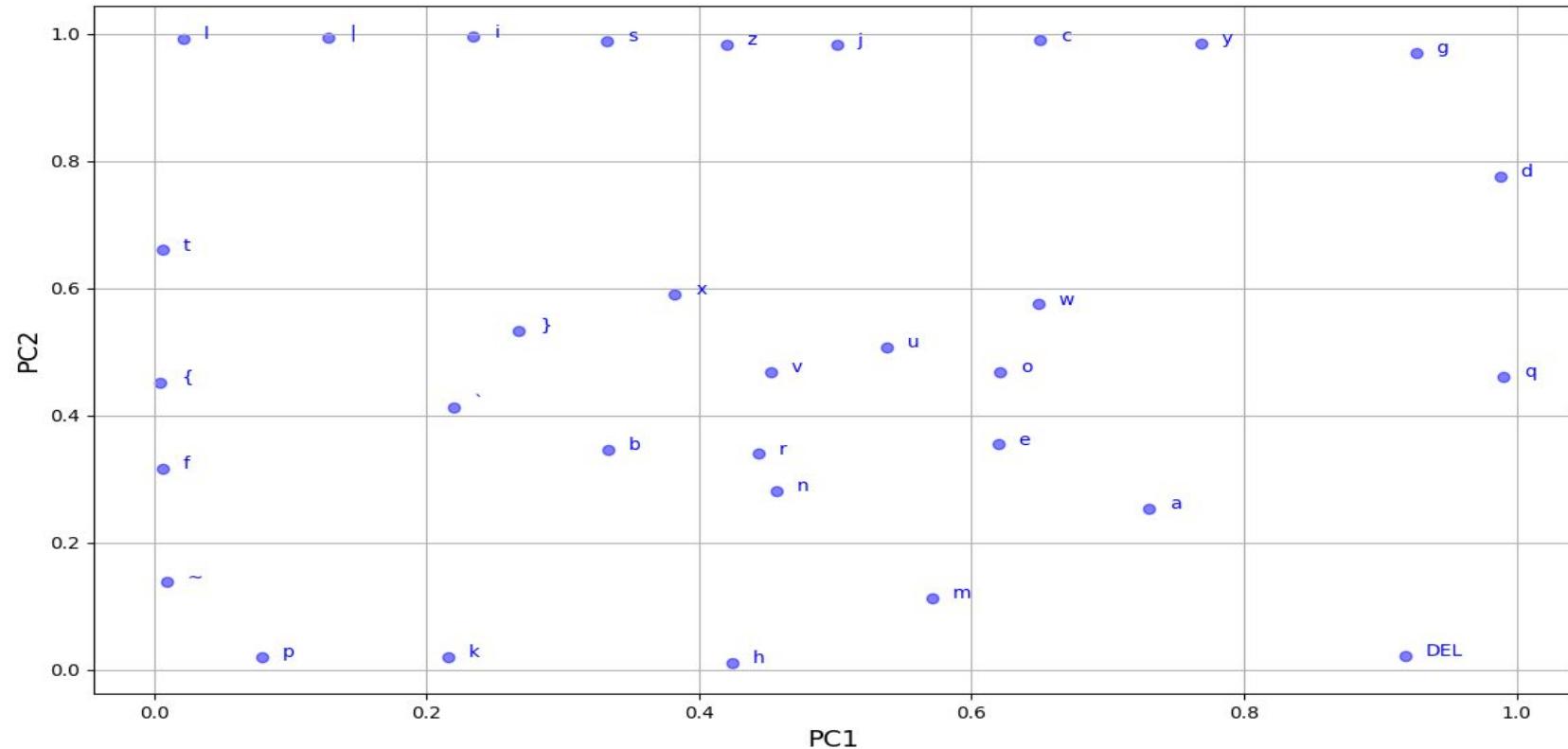


## Símbolos erróneos

- Un píxel: [ a, d, v, ], ~ ]
  - Dos píxeles: p

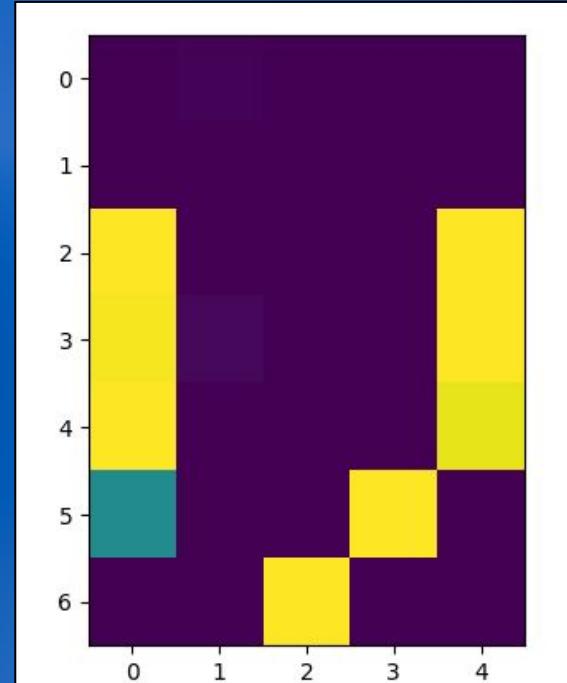
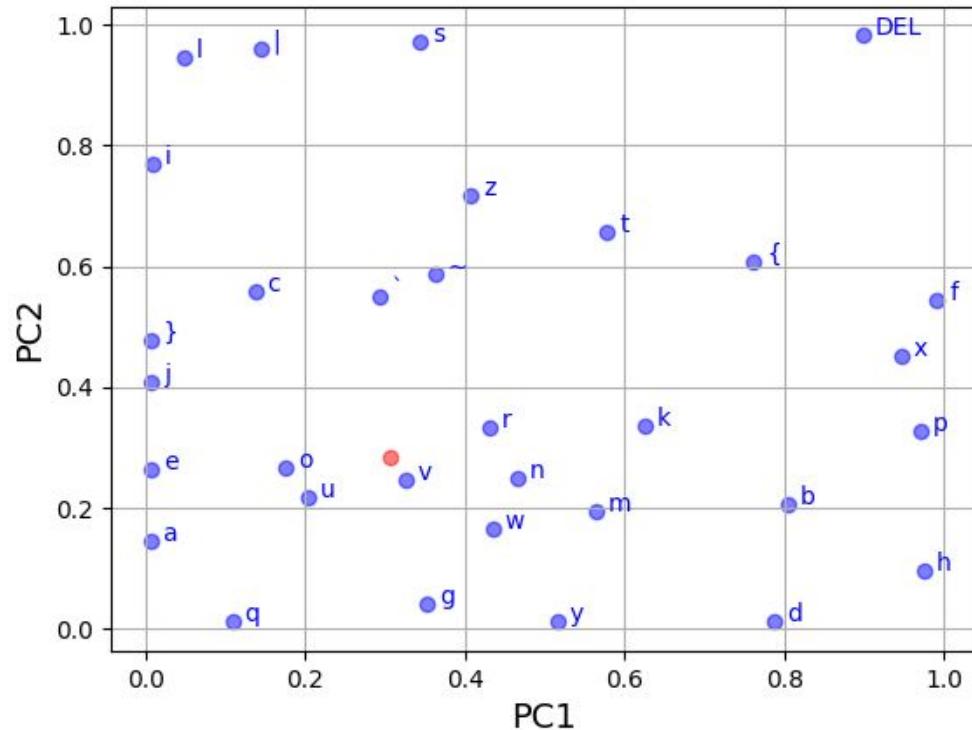
**Adaptativo: 20000 epochs |  $\eta_i = 0.003$  |  $\eta_f = 0.00075$  |  
Capas: [35, 28, 22, 17, 10, 2, 10, 17, 22, 28, 35]**

# Espacio latente



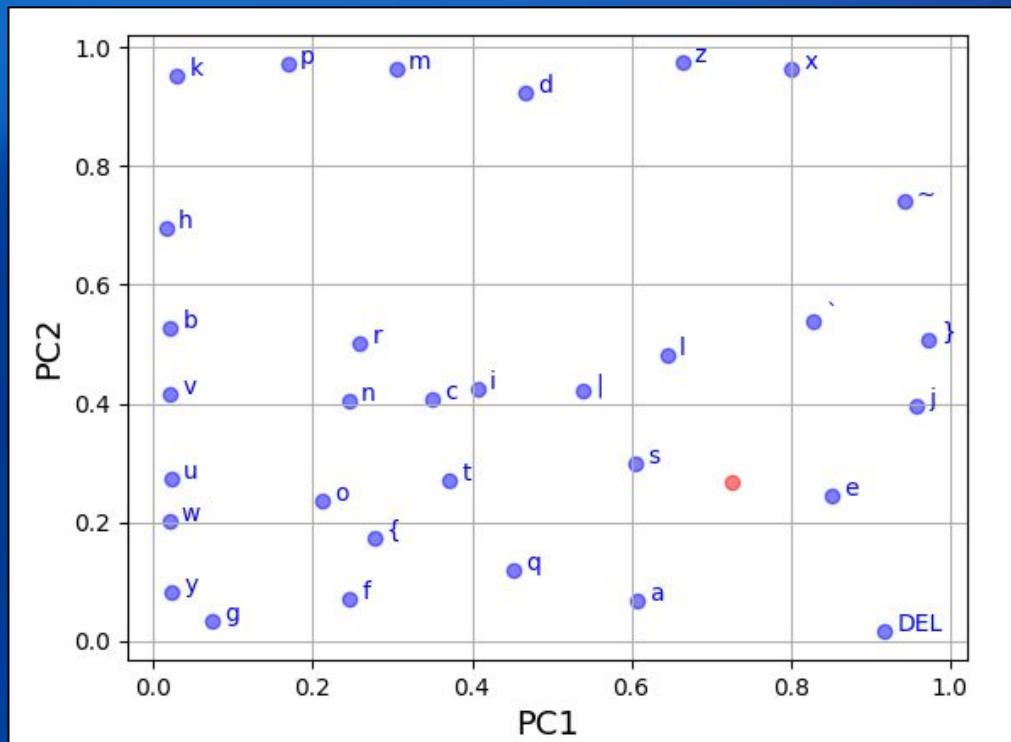
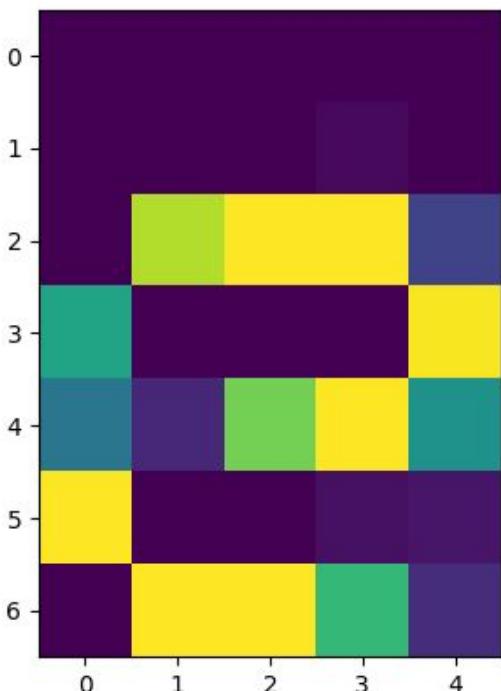
Adaptativo: 20000 epochs |  $\eta_i = 0.003$  |  $\eta_f = 0.00075$  |  
Capas: [35, 28, 22, 17, 10, 2, 10, 17, 22, 28, 35]

# Nuevos símbolos



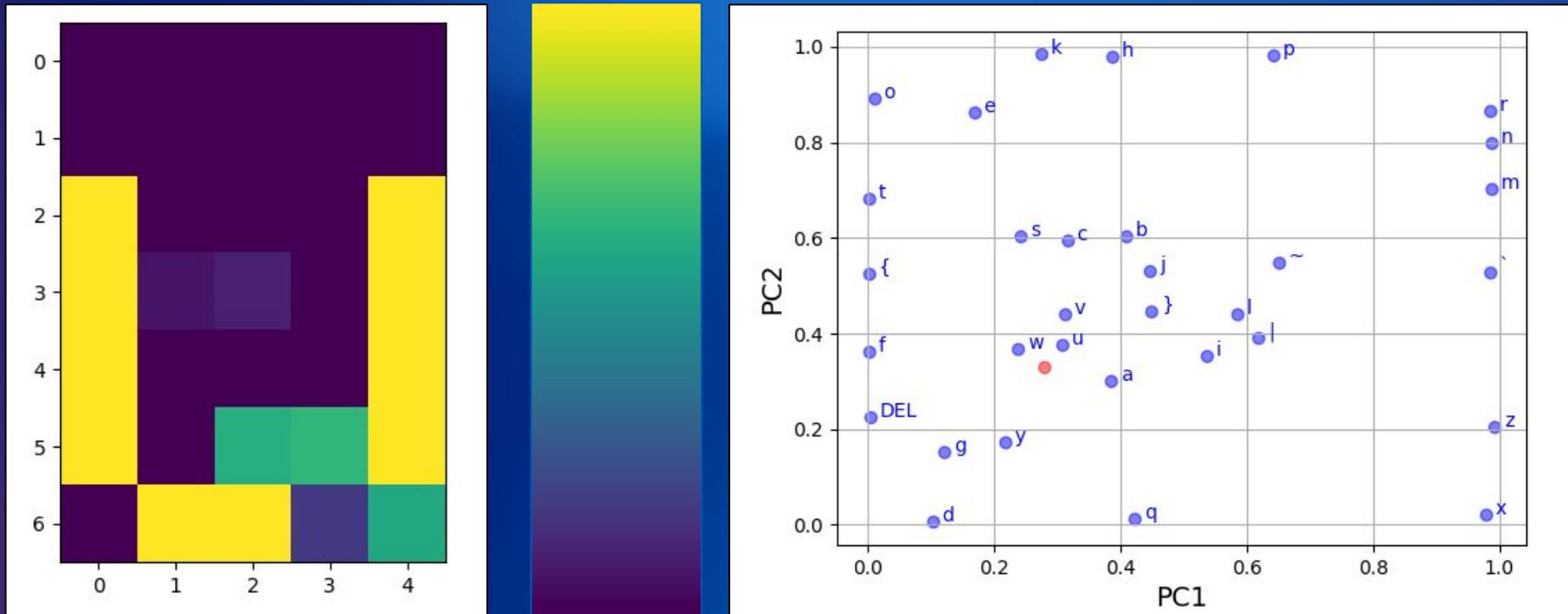
Adaptativo: 20000 epochs |  $\eta_i = 0.003$  |  $\eta_f = 0.00075$  |  
Capas: [35, 28, 22, 17, 10, 2, 10, 17, 22, 28, 35]

# Nuevos símbolos



Adaptativo: 20000 epochs |  $\eta_i = 0.003$  |  $\eta_f = 0.00075$  |  
Capas: [35, 28, 22, 17, 10, 2, 10, 17, 22, 28, 35]

# Nuevos símbolos



Adaptativo: 20000 epochs |  $\eta_i = 0.003$  |  $\eta_f = 0.00075$  |  
Capas: [35, 28, 22, 17, 10, 2, 10, 17, 22, 28, 35]

# Ej1\_b

## Denoising Autoencoder

- ❖ ***Ruido***
- ❖ ***Arquitectura***
- ❖ ***Denoising***



# Ruido

Se caracterizó el ruido mediante dos funciones distintas:

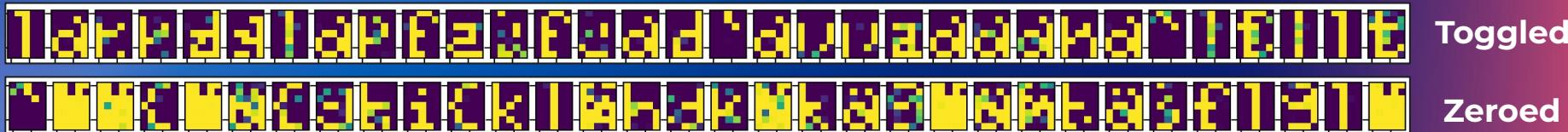
- **Alternando el pixel** (*Toggled*):  $v_f = 1 - v_i$
- **“Apagando” el pixel** (*Zeroed*):  $v_f = 0$

Luego, dado un porcentaje de ruido se calcula la cantidad de píxeles que se deben modificar sobre el total.

# Arquitectura

Ruido: 0.5

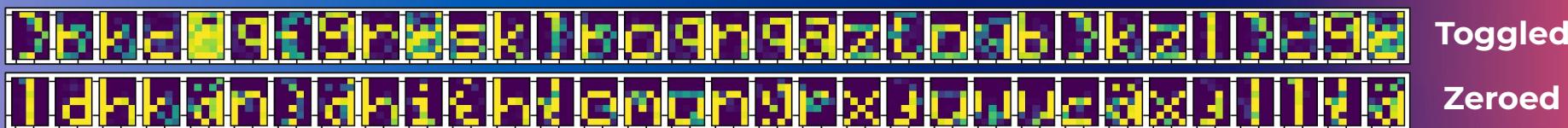
Capas: [35, 20, 2, 20, 35]



Capas: [35, 25, 20, 15, 2, 15, 20, 25, 35]



Capas: [35, 25, 20, 15, 10, 5, 2, 5, 10, 15, 20, 25, 30, 35]



30000 epochs

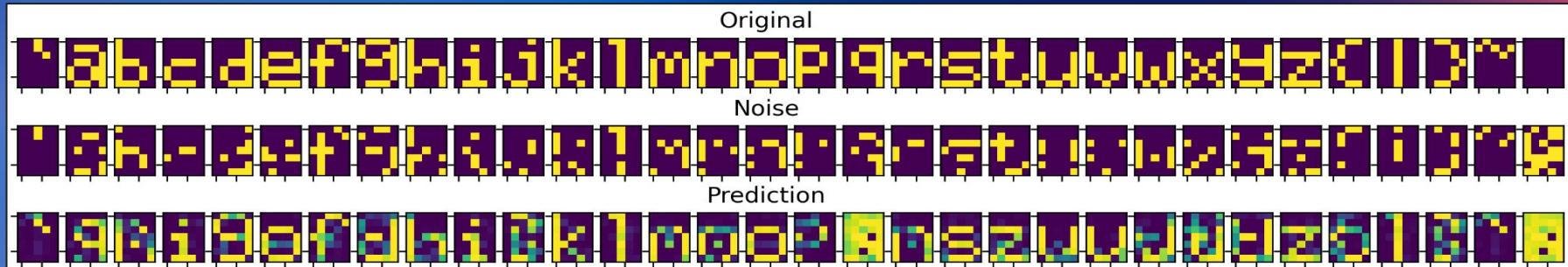
Adam:  $\beta_1=0.9$ ,  $\beta_2=0.999$ ,  
 $\epsilon=10^{-8}$

Adaptativo:  $\eta_i = 0.003$  |  $\eta_f = 0.00075$

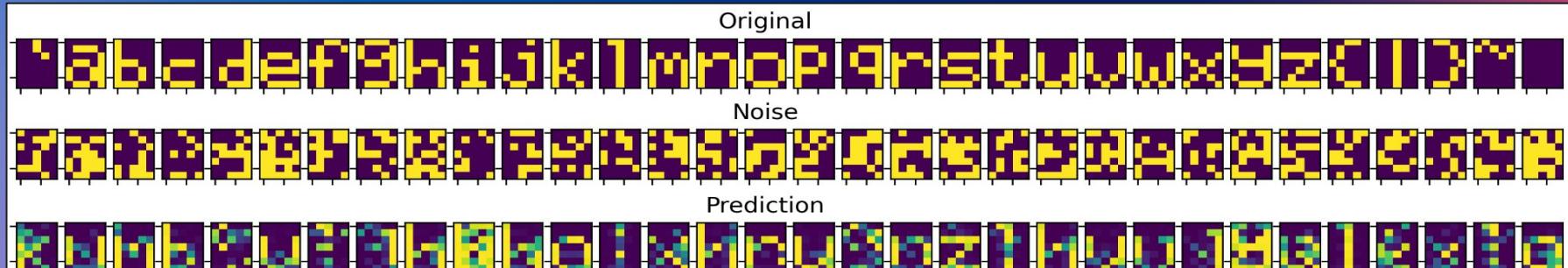
# Aumentando el espacio latente

Ruido: 0.5

Zeroed



Toggled



Capas intermedias: [30, 25, 20, 15]

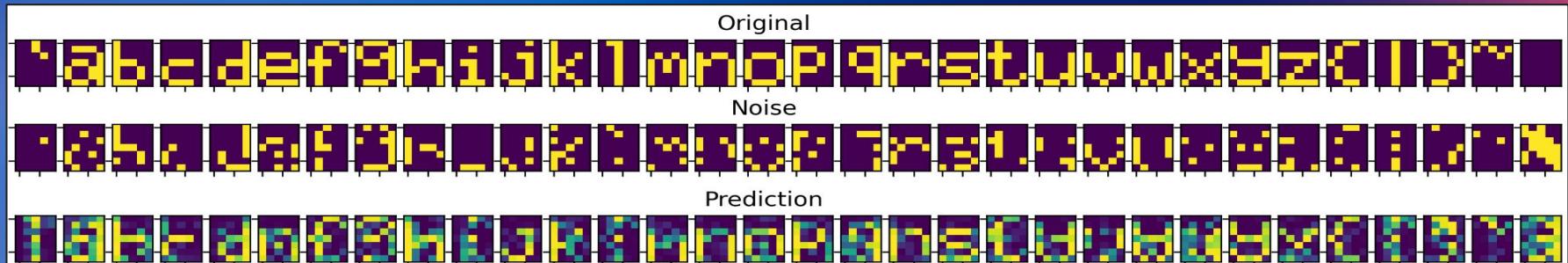
Espacio latente: 10

30000 epochs

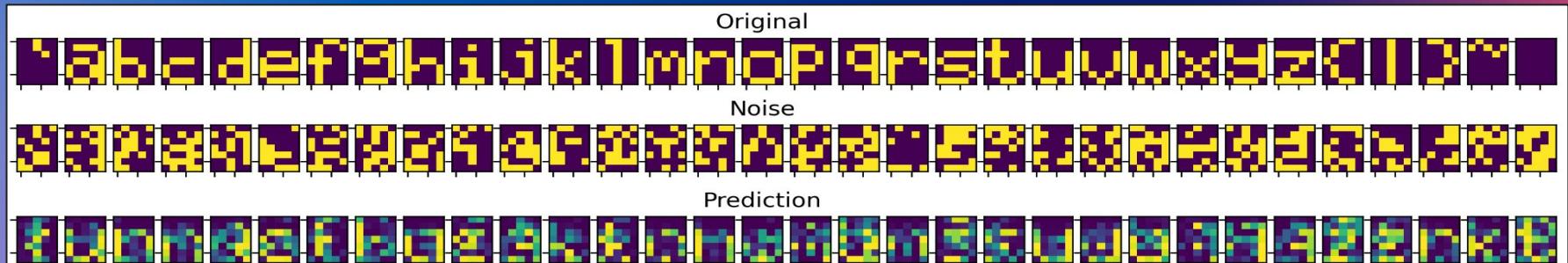
# Aumentando el espacio latente

Ruido: 0.5

Zeroed



Toggled



Capas intermedias: [30, 25]

Espacio latente: 20

30000 epochs

# Denoising

## Toggled

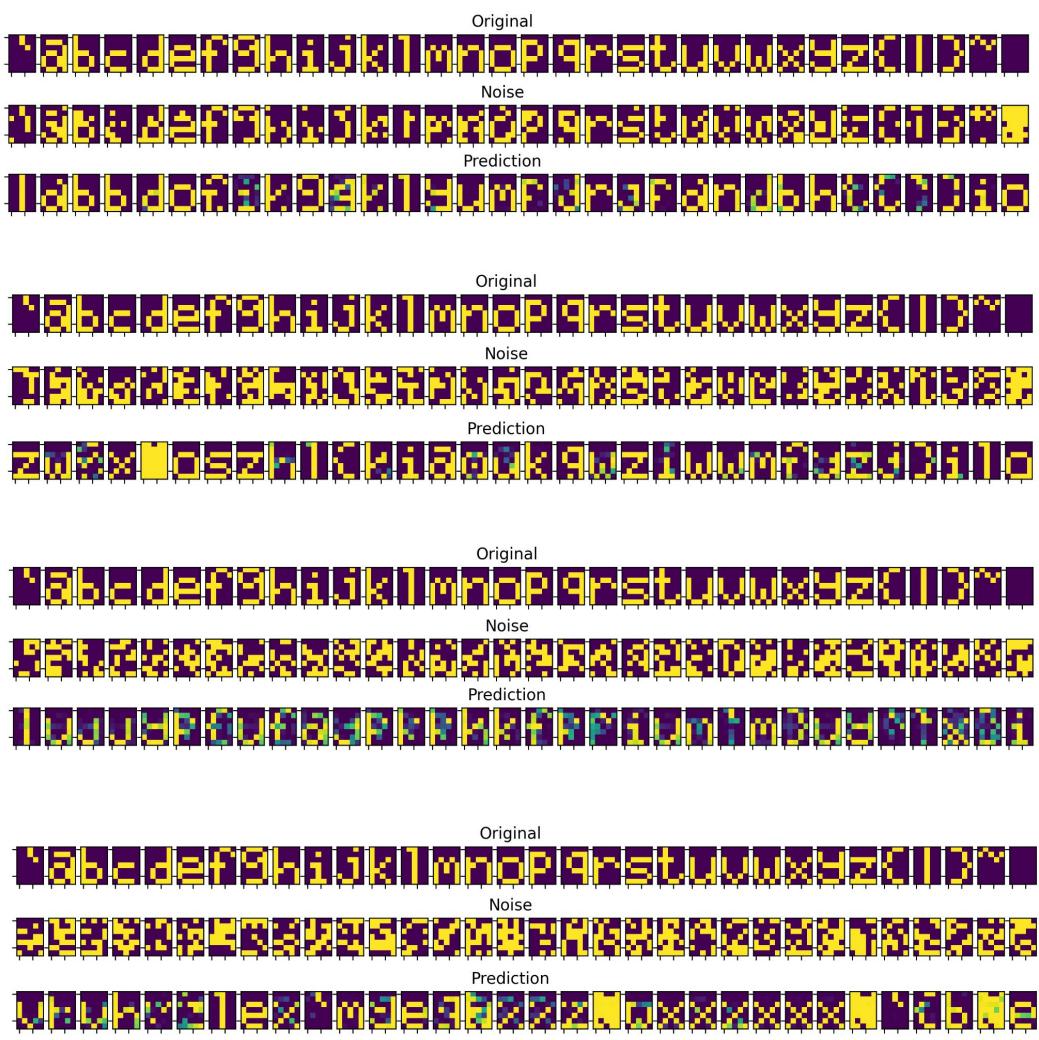
Noise pct: 0.1

Noise pct: 0.3

Noise pct: 0.6

Noise pct: 0.9

Adaptativo:  
30000 epochs  
 $\eta_i = 0.003$   
 $\eta_f = 0.00075$   
Capas: [30, 25, 20, 15, 10, 5]  
Latent Space = 2



# Denoising

## Zeroed

Noise pct: 0.1

Noise pct: 0.3

Noise pct: 0.6

Noise pct: 0.9

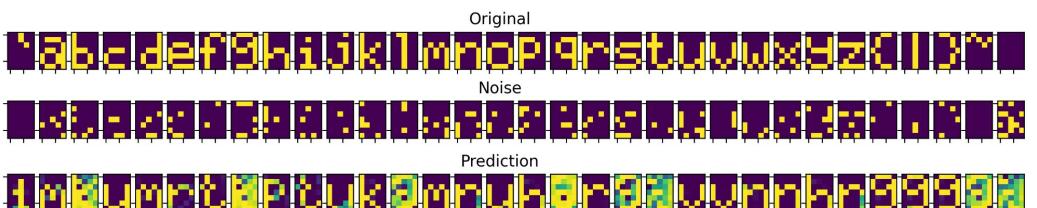
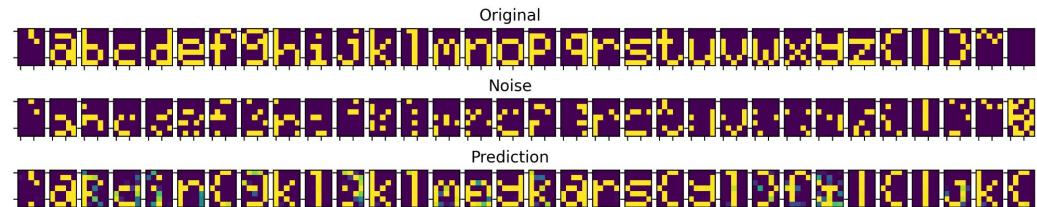
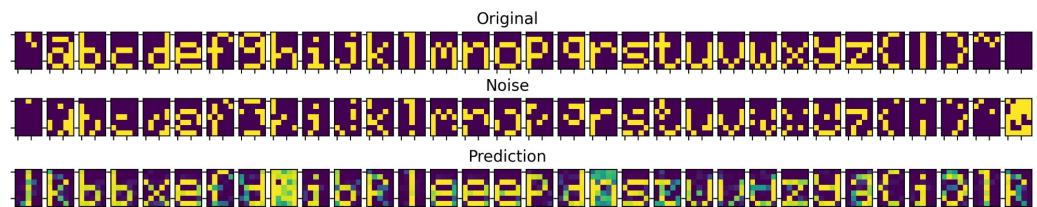
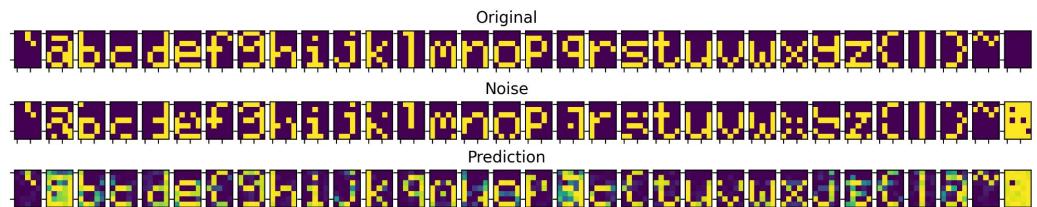
Adaptativo:  
30000 epochs

$\eta_i = 0.003$

$\eta_f = 0.00075$

Capas: [30, 25, 20, 15, 10, 5]

Latent Space = 2



# Ej2

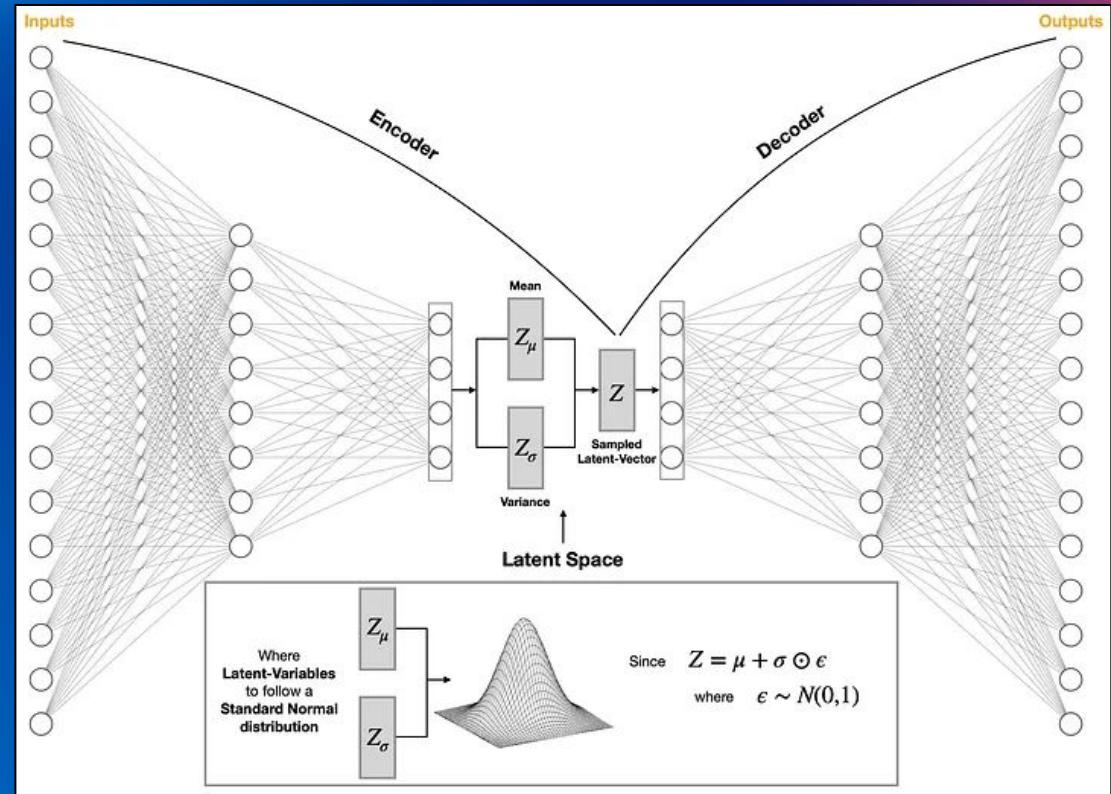
## Variational Autoencoder

- ❖ ***Arquitectura***
- ❖ ***Nuevo conjunto***
- ❖ ***VAE***

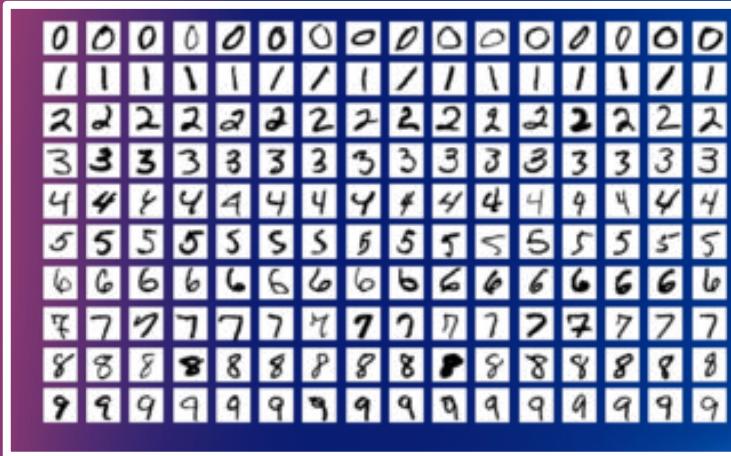


# Arquitectura

Se le agrega una capa estocástica de distribución gaussiana en el medio para generar nuevos samplings para pasarle al decoder.



# Nuevo conjunto (Mnist data set)



- Set de 26000 imágenes en formato bitmap de 28x28, representando 10 dígitos (0 a 9).
- Para pasarle el dataset al input, se normalizan los valores dividendo por el máximo tamaño de pixel (255)

# VAE

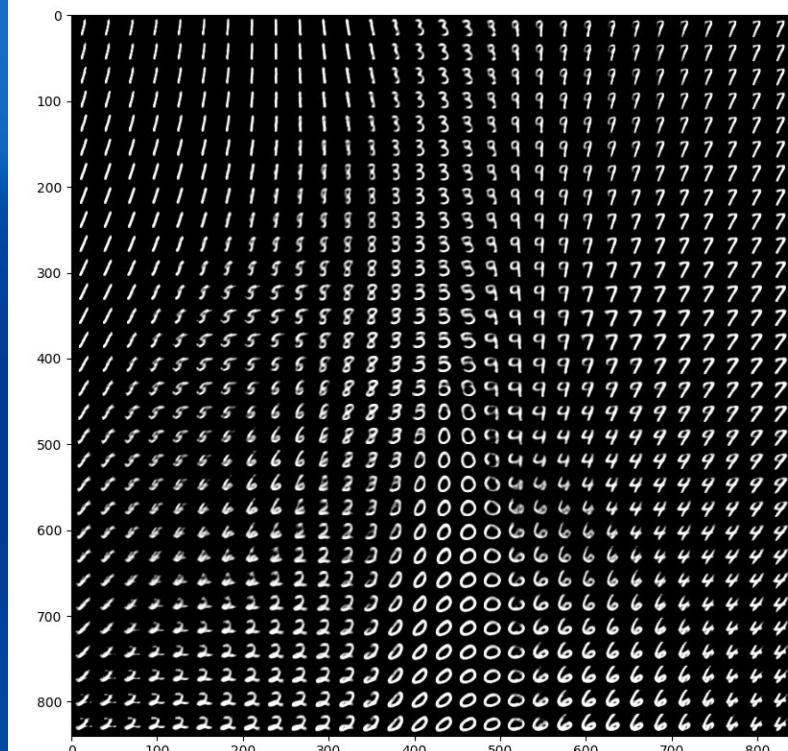
Prueba:

Arquitectura: [784, 64, 16, 8, 2, 8, 16, 64, 784]

Épocas: 1000

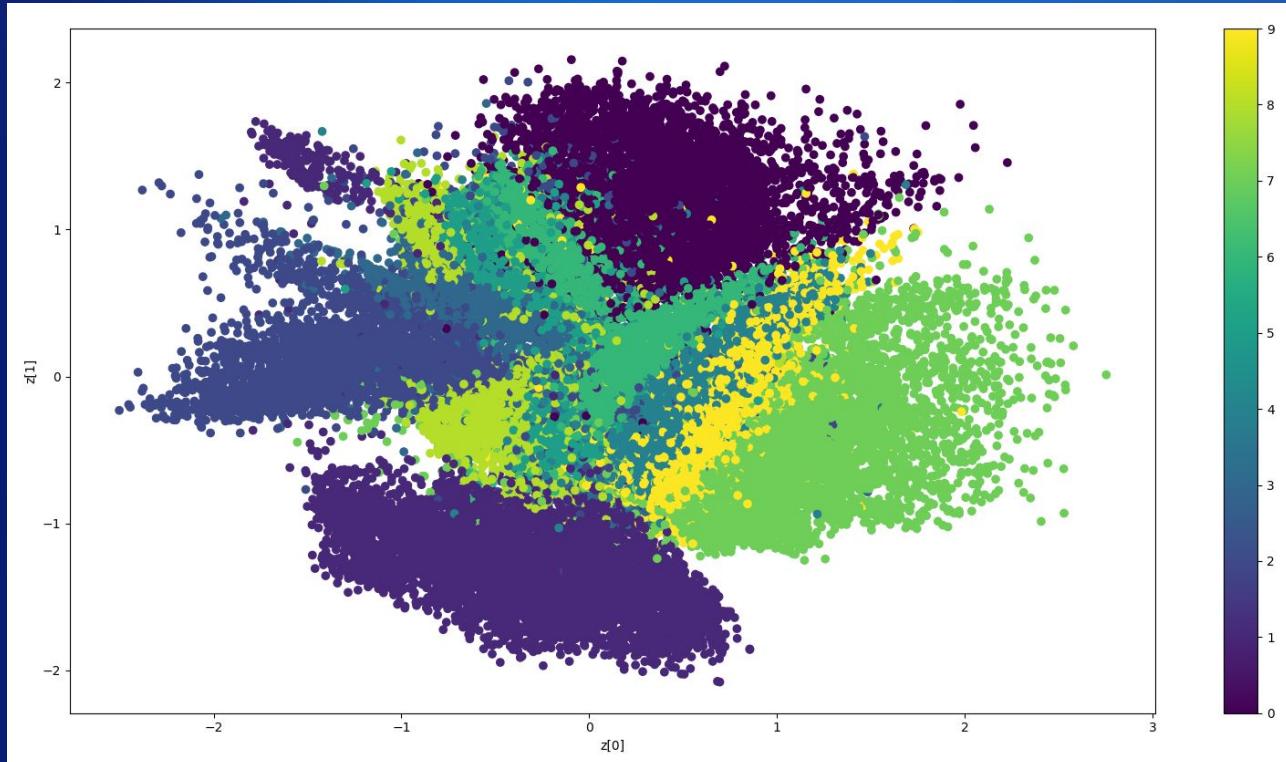
eta: 0.001

Método de optimización: Adam



n puntos de std [-1.5 1.5]

# Espacio latente



Loss se calcula como la media entre el MSE + KL loss

# Conclusiones

- ❖ El autoencoder logra aprender ciertos patrones mejor que otros.
- ❖ Es posible aprender el conjunto completo con un margen de error máximo de 1 pixel.
- ❖ Este modelo de autoencoder planteado es muy susceptible al ruido. A niveles muy altos de ruido aumenta la tasa de errores.
- ❖ El Autoencoder Variacional, a diferencia del Autoencoder normal, permite obtener resultados más similares entre los distintos puntos del espacio latente.

**¡GRACIAS!**