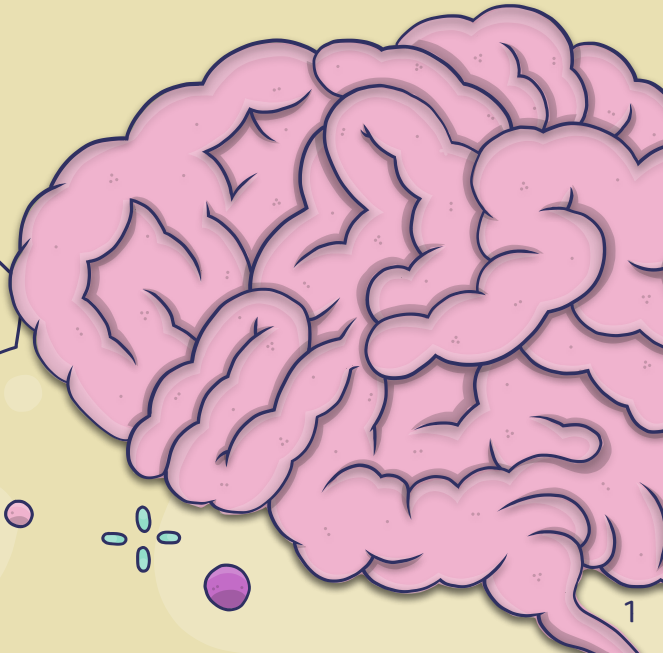
DEEP LEARNING



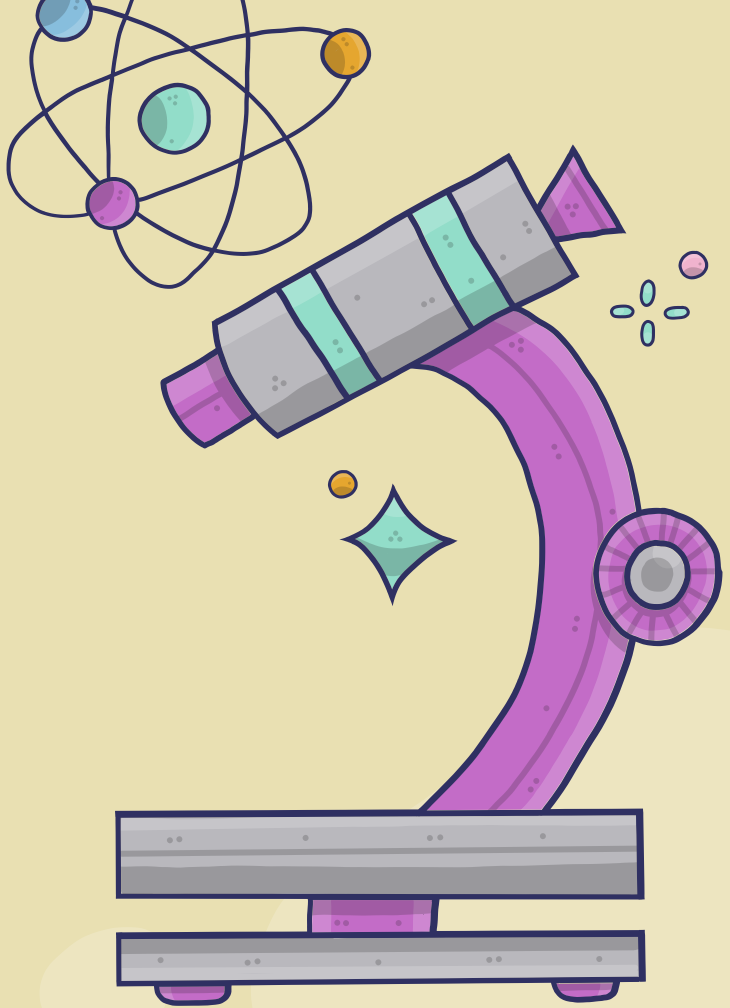
Sistemas de Inteligencia Artificial 2C-2022
Grupo 1 - "Augusta Ada King"

Domingues Paula	60148
Donikian Gastón	60067
Pavan Matias	58296
Rodriguez Manuel	60258

- Autoencoder generando carácteres
- Denoising autoencoder sobre carácteres
- Autoencoder variacional sobre pokemones

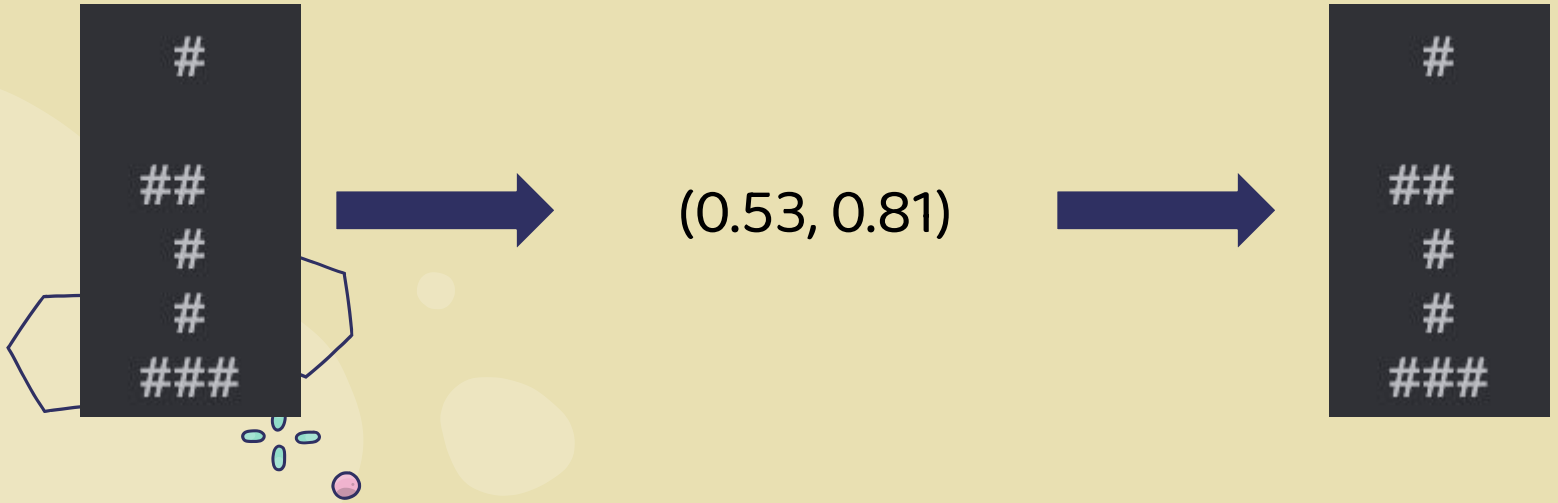
01

Autoencoder



Introducción

Busca aprender a reducir la dimensionalidad de los inputs
de forma que al volver a la dimensión
original se obtenga el input

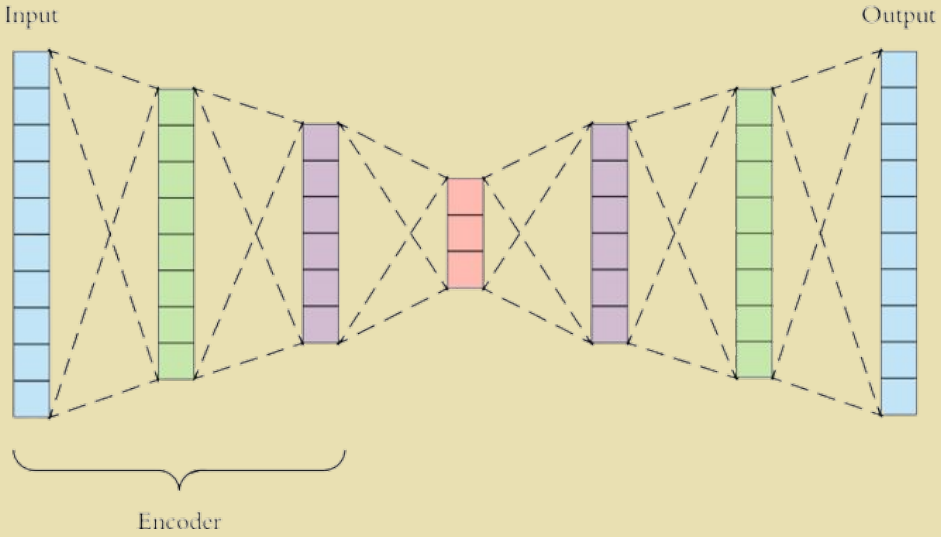


1.1

Arquitectura de un Autoencoder

1

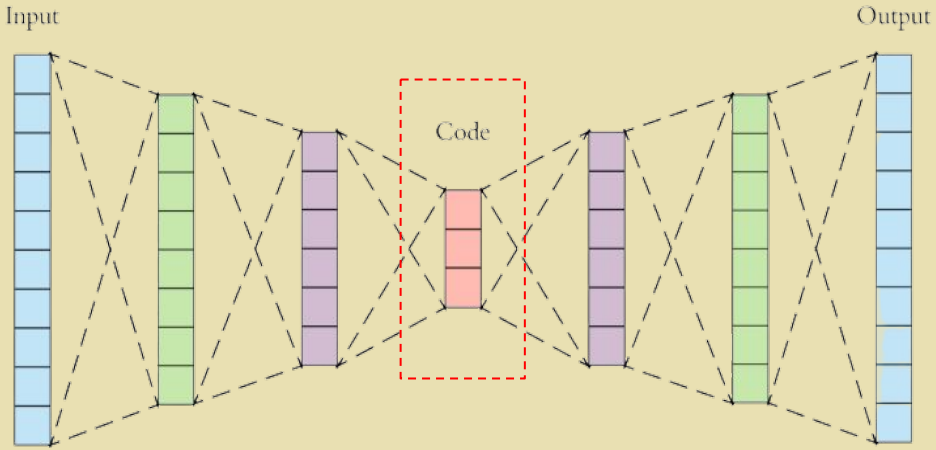
Encoder



Procesa los inputs y reduce la dimensión de la entrada al tamaño del espacio latente

2

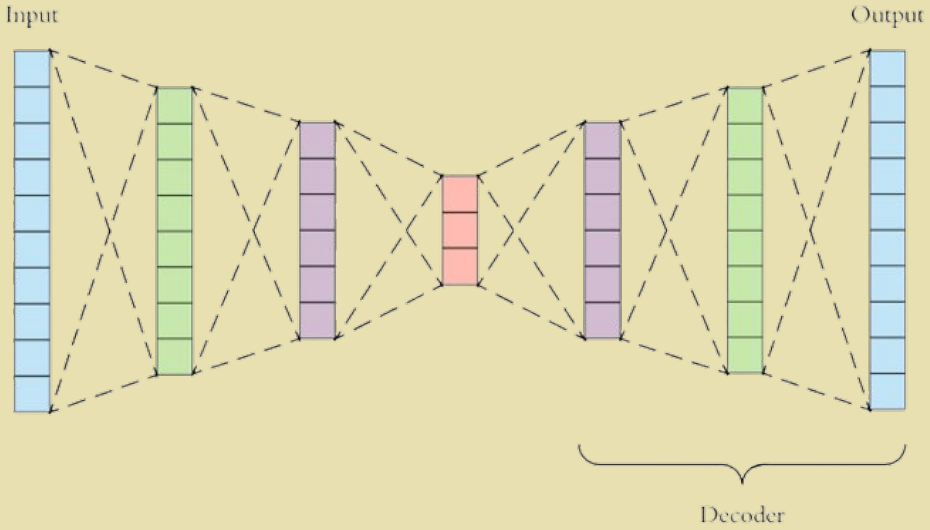
Espacio Latente



Es la reducción de dimensionalidad que estamos buscando, la red es simétrica sobre esta capa

3

Decoder

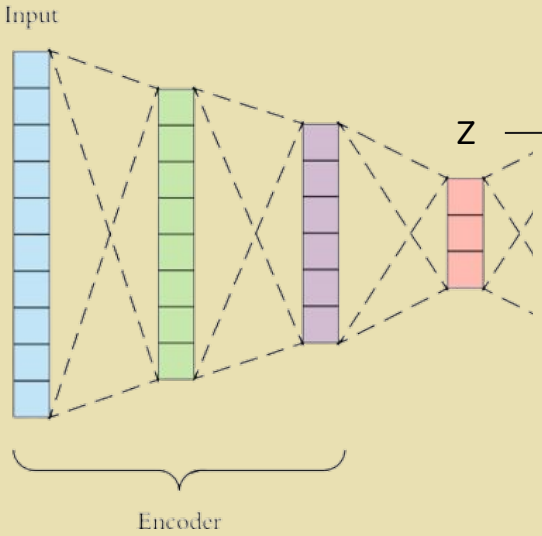


Descomprime los datos y los retorna a su dimensión original

1.2

Activaciones

Activación Encoder



$$z = \sigma(Wx + b)$$

z : valor en el espacio latente

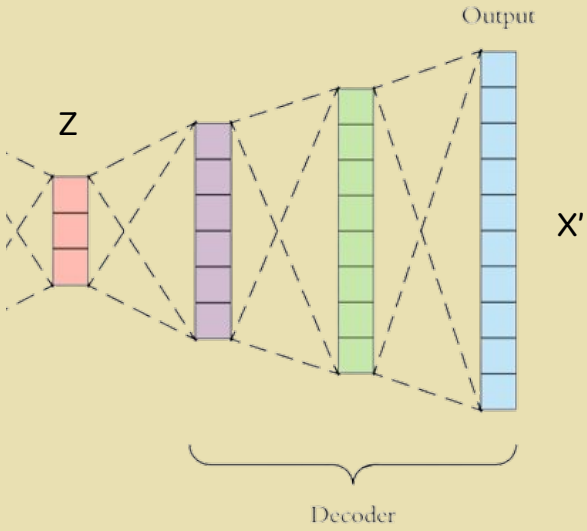
σ : funcion de activacion

W : vector de pesos

x : valores de entrada

b : bias

Activación Decoder



$$x' = \sigma'(W'z + b')$$

z : valor en el espacio latente

σ' : funcion de activacion

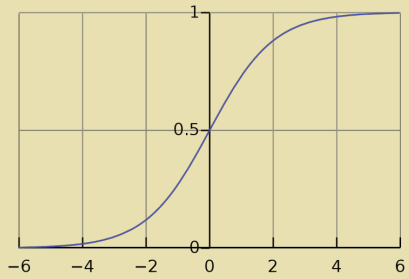
W' : vector de pesos

x' : valores de salida

b' : bias

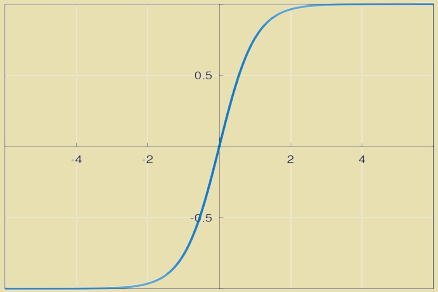
Funciones de activación

No lineal - Sigmoidal - Logística



$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

Tangente Hiperbólica



$$tanh(x) = \frac{sinh(x)}{cosh(x)}$$

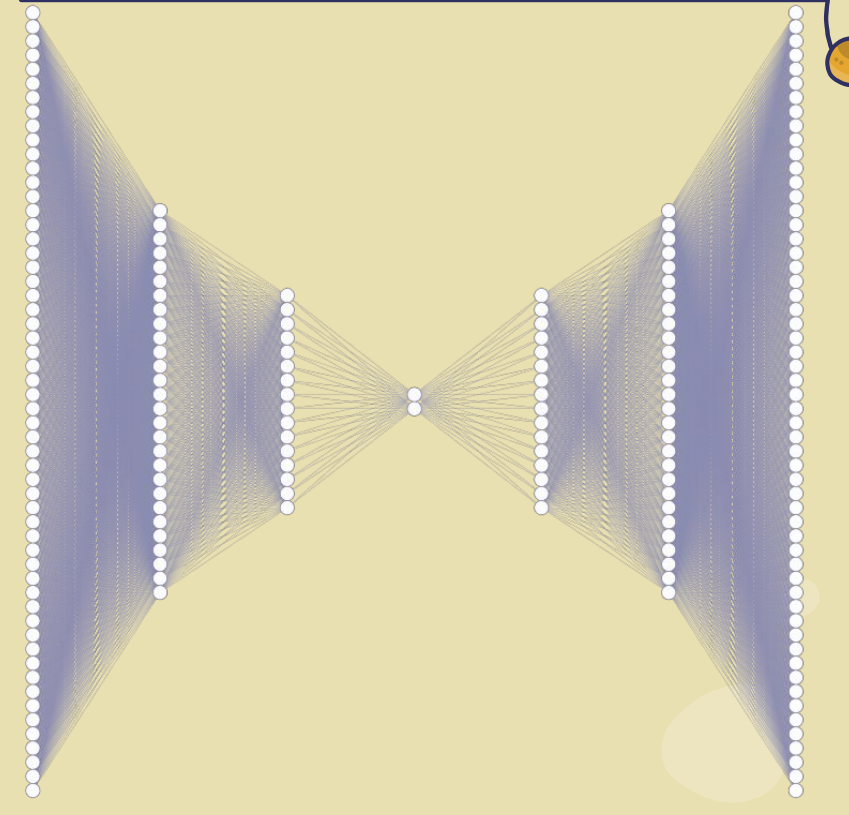
Retropropagación

$$\iota(x, x') = ||x - x'||^2 = ||x - \sigma'(W'(\sigma(Wx + b)) + b')||^2$$

Se busca minimizar esta función de "pérdida"

1. Se toman todos los pesos de la iteración actual
2. Calculamos la activación de la entrada y la comparamos con el valor esperado que le corresponde
3. Realizamos gradiente descendiente sobre los pesos de la iteración actual, actualizandolos
4. Repetimos desde el primer paso con la siguiente iteración

Arquitectura planteada



Capas intermedias: 28-16-2-16-28

1.3

Dataset

Propiedades del dataset

- A partir de fonts.h, se generan matrices binarias de tamaño 7*8
- Carácter “a” en fonts.h es: [0x00, 0x0e, 0x01, 0x0d, 0x13, 0x13, 0x0d], pero luego de la transformación a matriz se visualizará como:

```
00000000
00001110
00000001
00001101
00010011
00010011
00001101
```

- El dataset se encuentra compuesto por 32 carácteres

1.4

Aprendizaje de fuente

Objetivos

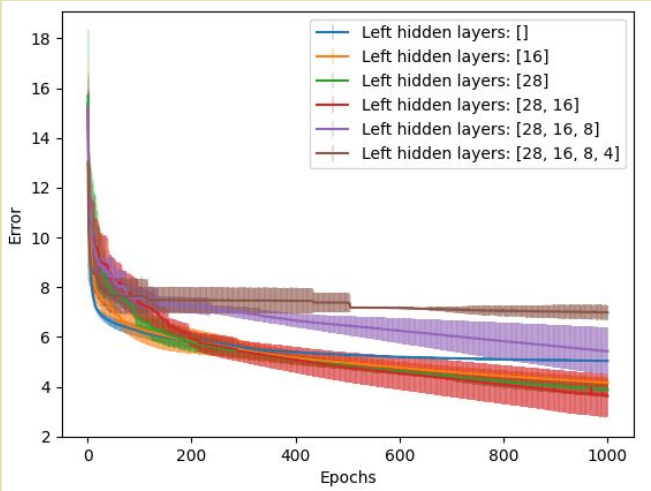
- Evaluar como el autoencoder aprende el dataset y aplicar mejoras

- Generar nuevos carácteres que no pertenezcan al dataset de entrenamiento

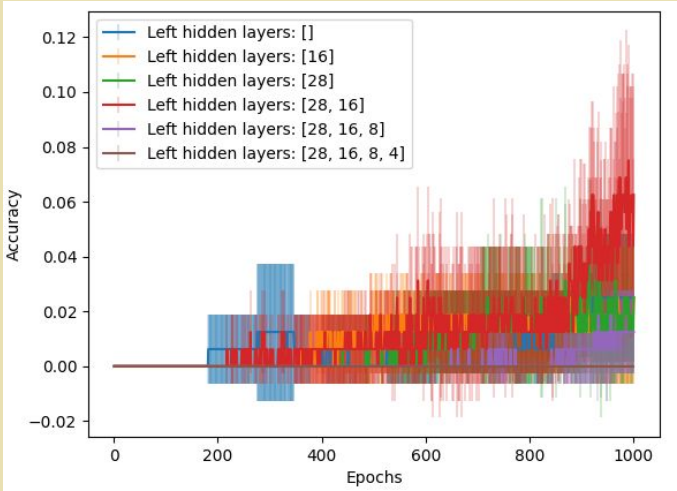
Aprendizaje de fuente
Resultados variando capas

Parámetros:
n = 0.05
Epochs = 1000
Ejecuciones = 5
F. de activación = Logística
Momentum = 0.3
Con adam

Error

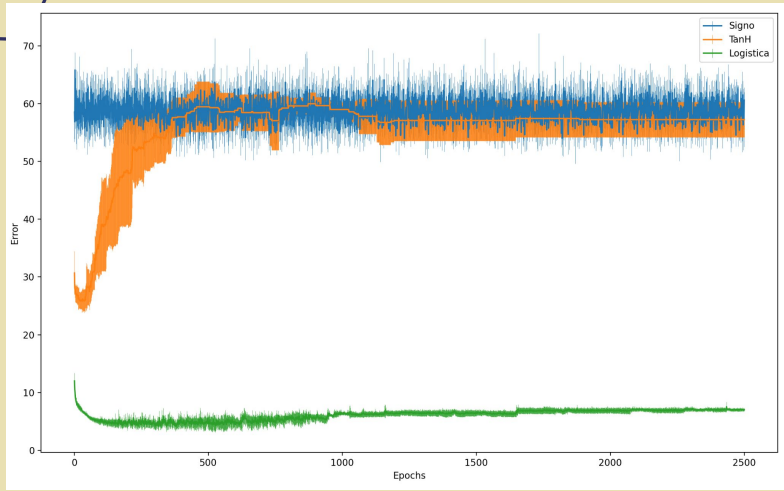


Accuracy

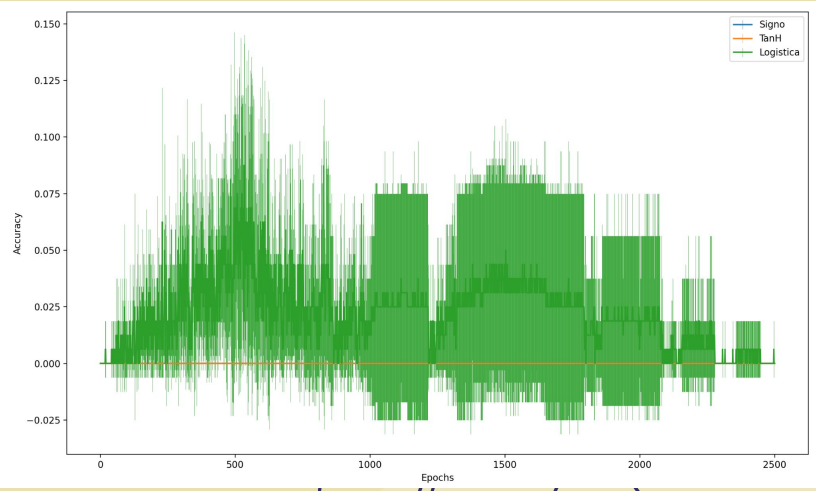


Aprendizaje de fuente
Resultados variando F. de activación

Error



Accuracy



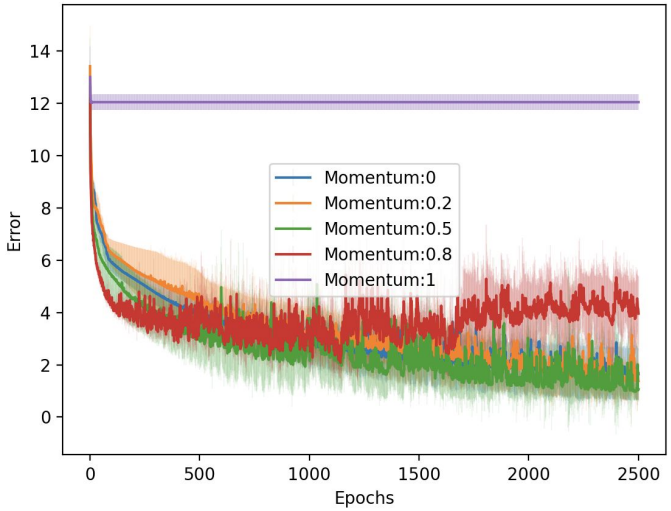
Parámetros:
n = 0.05
Epochs = 1000
Ejecuciones = 5
Capas = [28,16,2,16,28]
Sin adam
Sin momentum

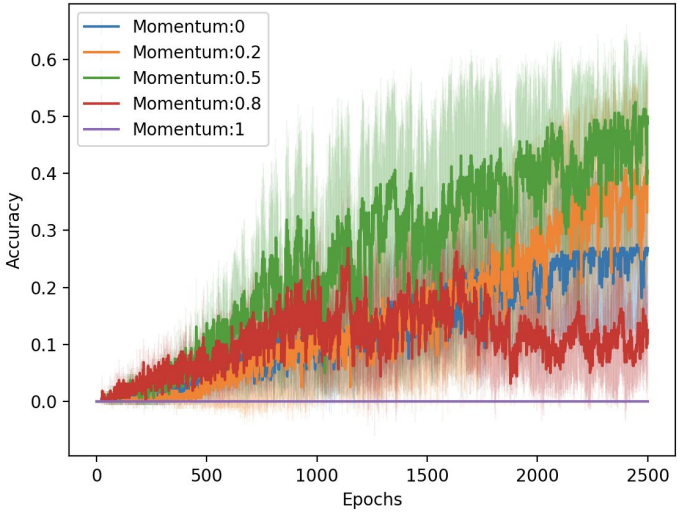
Aprendizaje de fuente
Resultados variando momentum sin Adam

Error

Accuracy

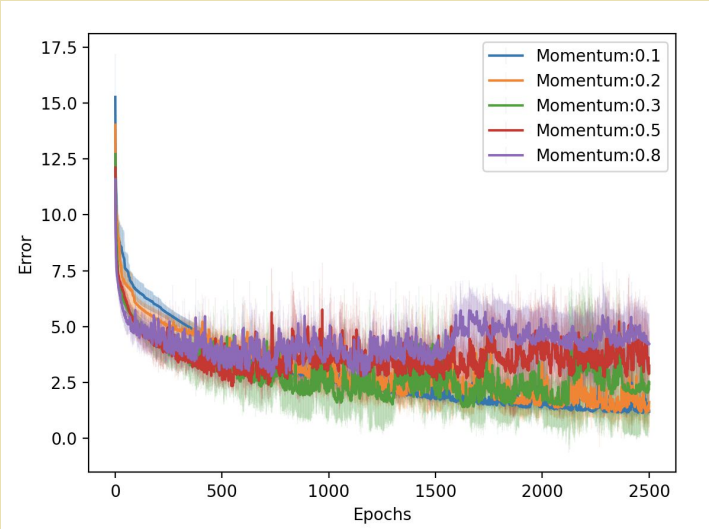
Parámetros:
n = 0.05
Epochs = 1000
Ejecuciones = 5
F. de activación = Logística
Capas = [28,16,2,16,28]
Sin Adam



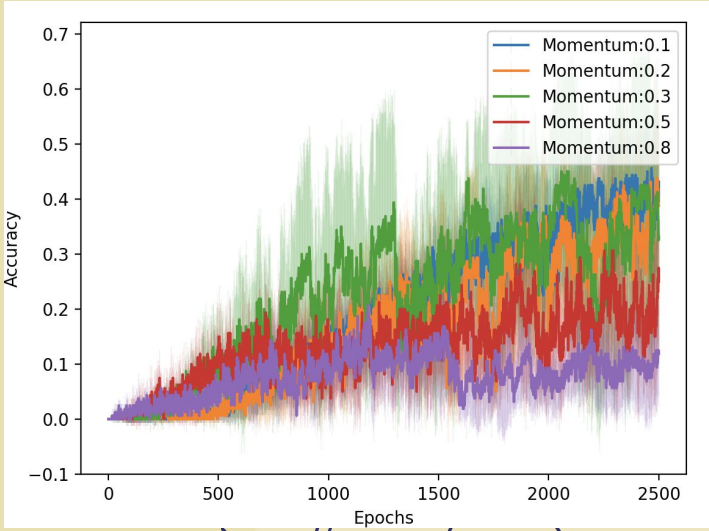


Aprendizaje de fuente
Resultados variando momentum con Adam

Error



Accuracy

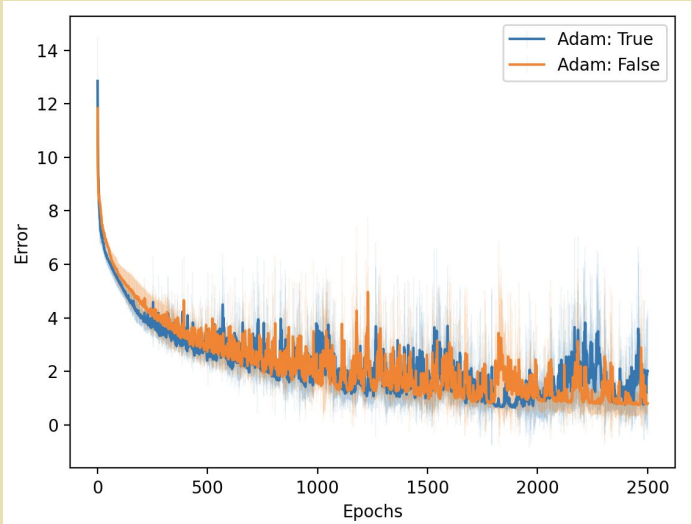


Parámetros:
n = 0.05
Epochs = 1000
Ejecuciones = 5
F. de activación = Logistic
Capas = [28,16,2,16,28]
Con Adam

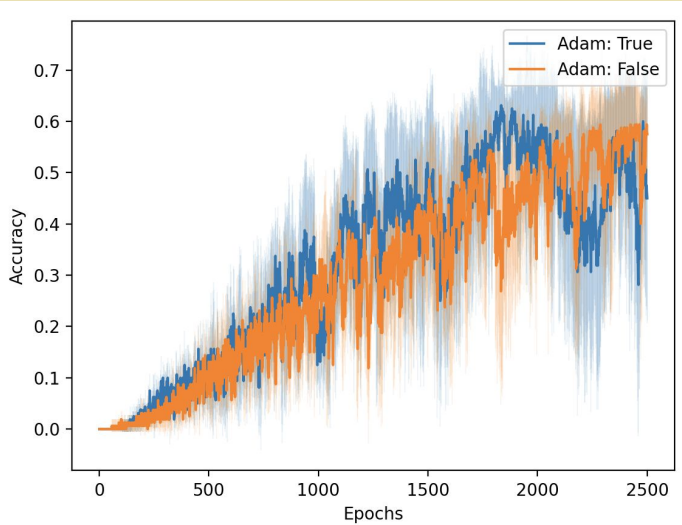
Aprendizaje de fuente
Resultados con y sin adam

Parámetros:
n = 0.05
Epochs = 1000
Cantidad de ejecuciones = 5
F. de activación = Logistic
Capas = [28,16,2,16,28]
Sin adam: Momentum = 0.5
Con adam: Momentum = 0.3

Error



Accuracy



Aprendizaje de fuente
Definición de adaptative learning rate

Pseudocódigo

β = 0.03

α = 0.01

Si el error es consistente hace más de 3 pasos:

Si el error está decrementando:

$$\eta = \eta + \alpha * \eta$$

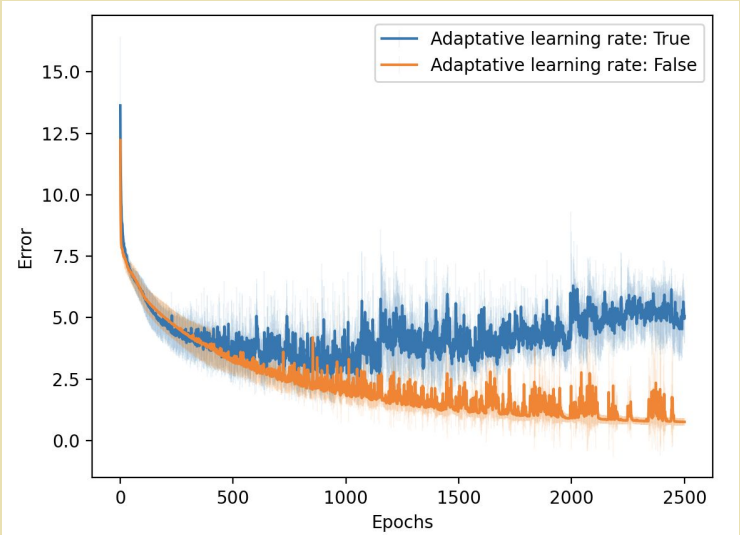
Si el error está incrementando:

$$\eta = \eta - \beta * \eta$$

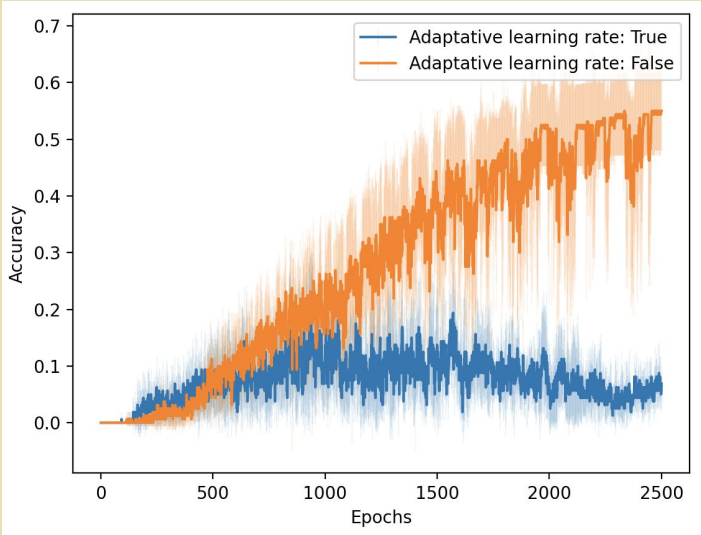
Aprendizaje de fuente
Resultados con y sin adaptative learning rate

Parámetros:
n = 0.05
Epochs = 1000
Ejecuciones = 5
F. de activación = Logistic
Capas = [28,16,2,16,28]
Momentum = 0.5
Adam = False

Error

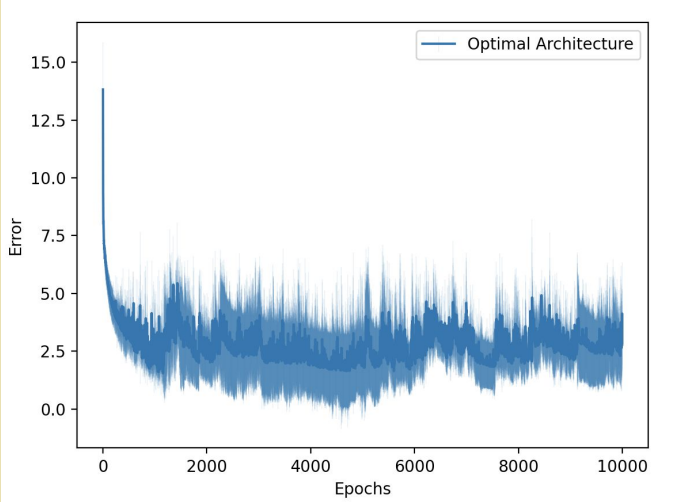


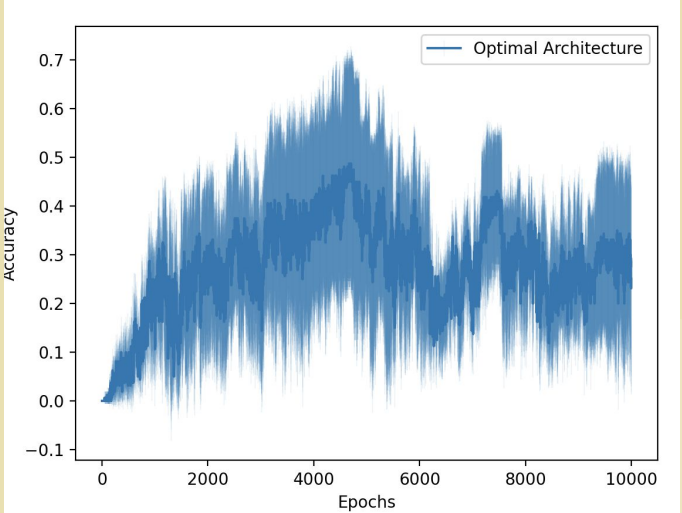
Accuracy



Aprendizaje de fuente
Arquitectura ganadora

Parámetros:
n = 0.05
Epochs = 10000
Cantidad de ejecuciones = 5
F. de activación = Logística
Momentum = 0.3
Con adam
Capas = [28,16,2,16,28]





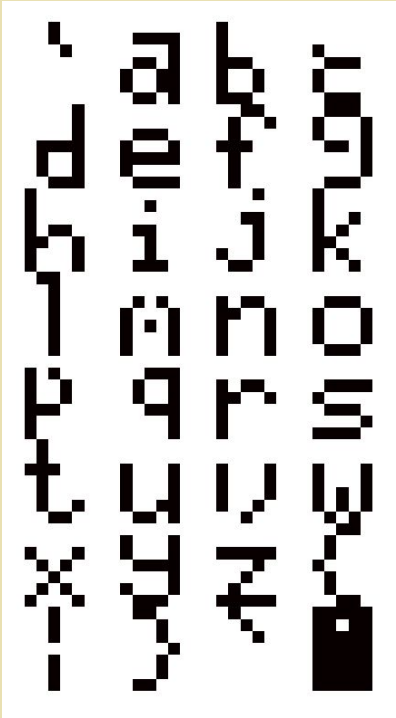
Máximo accuracy = 0.72 (23/32 carácteres)

No logró aprender el dataset en su completitud por la gran cantidad de datos y baja dimensión del espacio latente.

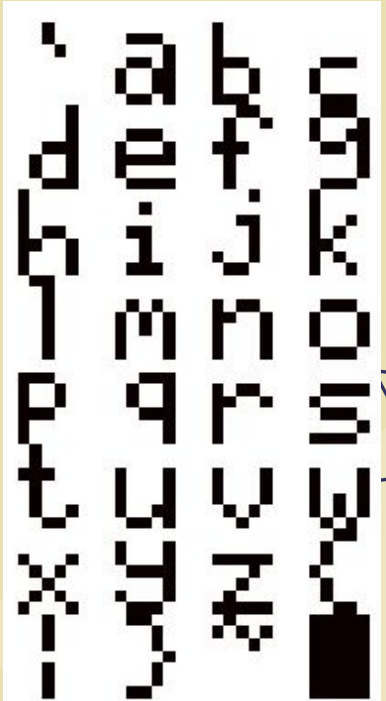
Aprendizaje de fuente
Comparativa fuente resultante

Parámetros:
n = 0.05
Epochs = 1000
Ejecuciones = 5
F. de activación = Logística
Capas = [28,16,2,16,28]
Con Adam
Momentum = 0.3

Fuente resultante

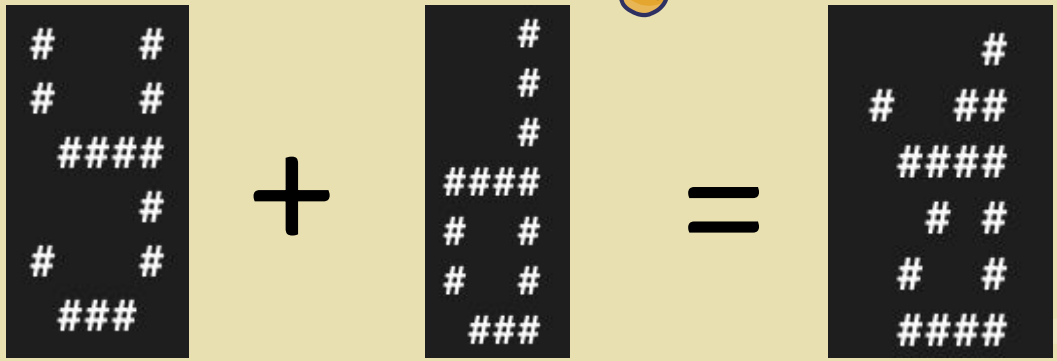


Fuente original



Aprendizaje de fuente
Nuevos caracteres

Parámetros:
n = 0.05
Epochs = 10000
Cantidad de ejecuciones = 5
F. de activación = Logística
Momentum = 0.3
Con adam
Capas = [28,16,2,16,28]

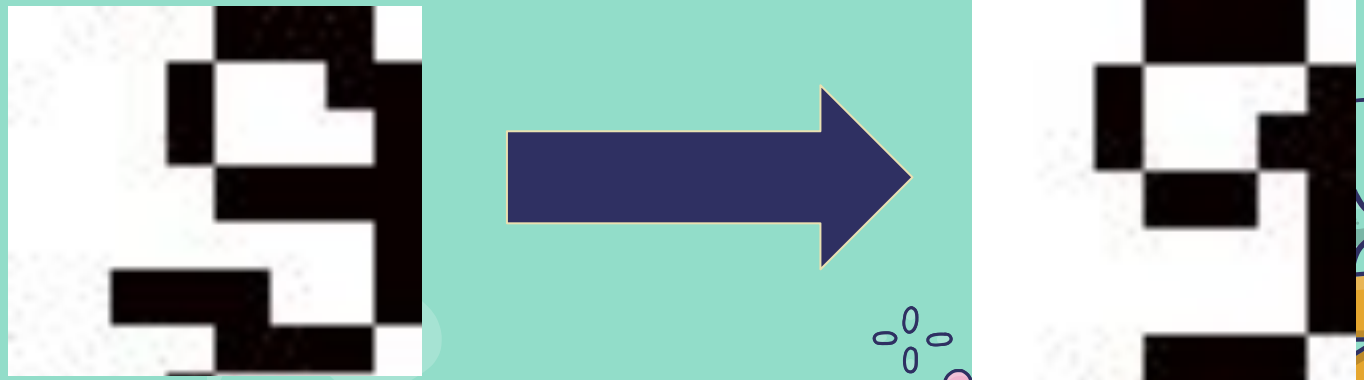


1.5

Aprendizaje de eliminación de ruido

Eliminación de ruido
Objetivo

- Evaluar como el autoencoder aprende a borrar ruido del dataset.
- En el entrenamiento al tomar una letra, se le aplica el ruido correspondiente y se compara el resultado con la original.



Eliminación de ruido
Explicación de ruido gaussiano

Pseudocódigo para generar ruido gaussiano en entrenamiento, sea noise_factor ∈ [0,1]:

```
Por cada época de entrenamiento:
    X = generar_dataset_ruidoso(X_train)
    ... // entrenamiento

generar_dataset_ruidoso(X):
    answer = copy(X)
    Por cada imagen en answer:
        Por cada pixel en imagen:
            pixel = pixel + noise_factor * random_gaussiano( μ=0, σ=1)
            Asegurarse pixel ∈ [0,1]
    return answer
```

Eliminación de ruido
Explicación de Salt and Pepper

Pseudocódigo para generar ruido Salt and Pepper en entrenamiento, sea *noise_factor* ∈ [0,1] arbitrario:

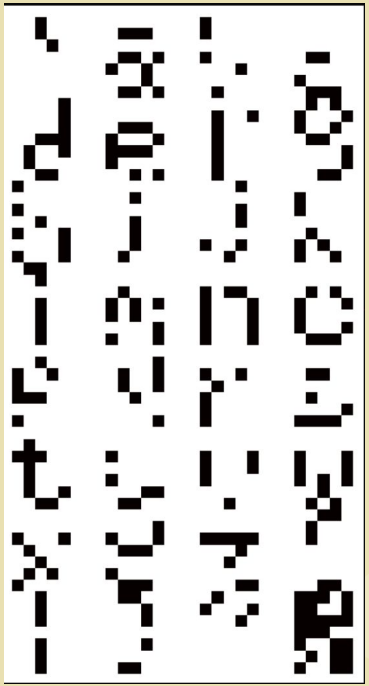
```
Por cada época de entrenamiento:
    X = generar_dataset_ruidoso(X_train)
    ... // entrenamiento

generar_dataset_ruidoso(X):
    answer = copy(X)
    Por cada imagen en answer:
        Por cada pixel en imagen:
            rand = Generar random ∈ [0,1]
            Si rand < noise_factor
                Swapear pixel de blanco a negro
                o de negro a blanco
    return answer
```

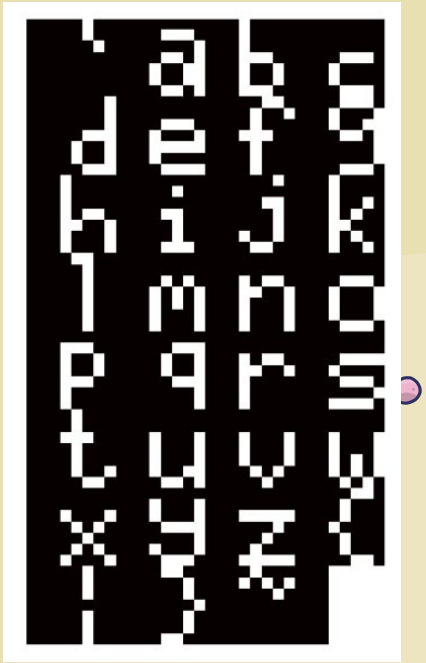
Tipos de Ruido:

Gaussiano:

Salt

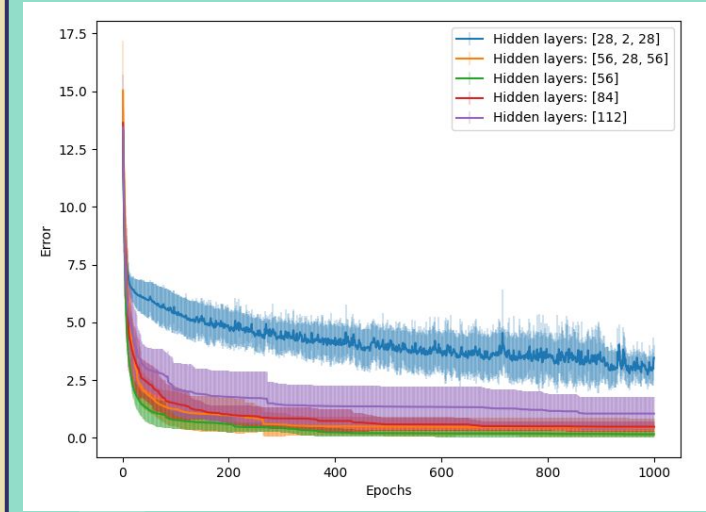


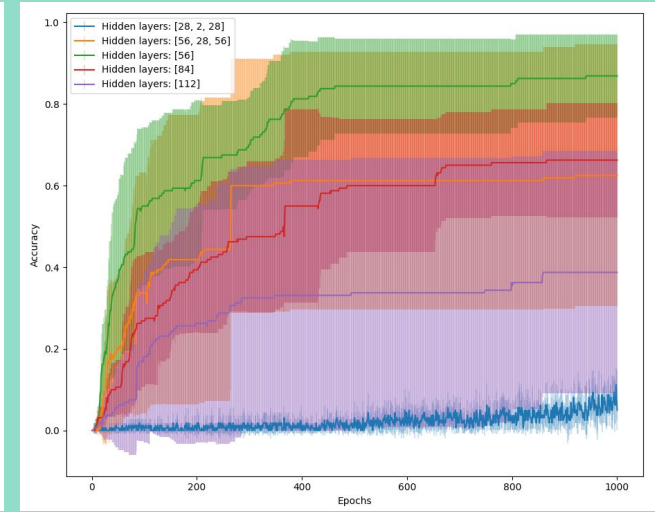
Salt & Pepper



Eliminación de ruido gaussiano
Resultados variando capas

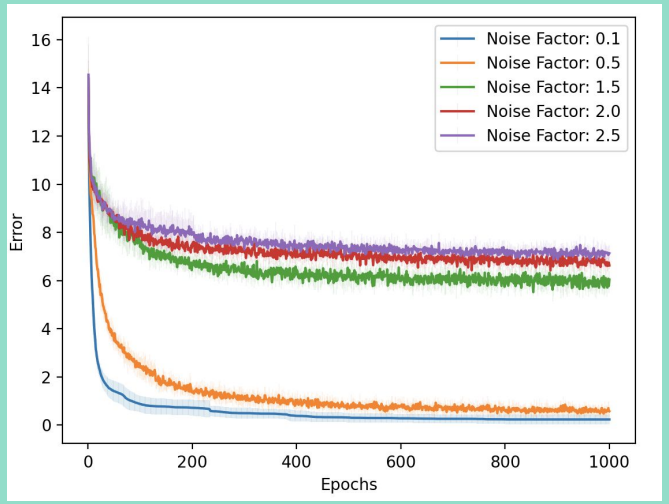
Parámetros:
n = 0.05
Epochs = 1000
Cantidad de ejecuciones = 5
F. de activación = Logística
Ruido = 0.1 (Gaussiano)
Con adam y momentum = 0.3

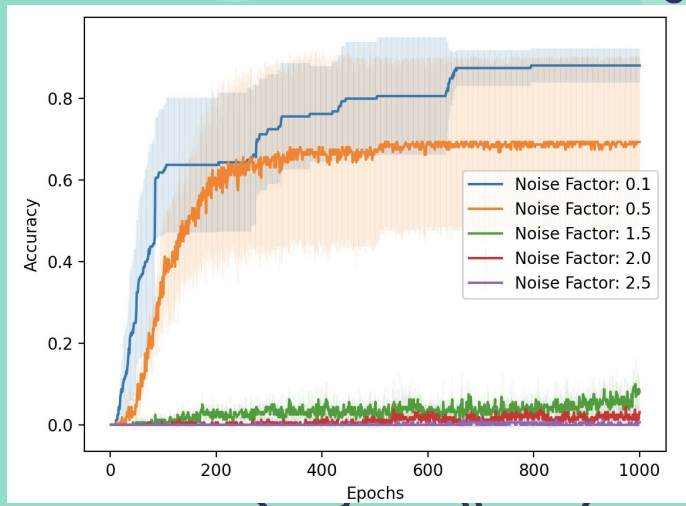




Eliminación de ruido gaussiano
Evolución de error y accuracy variando factor de ruido

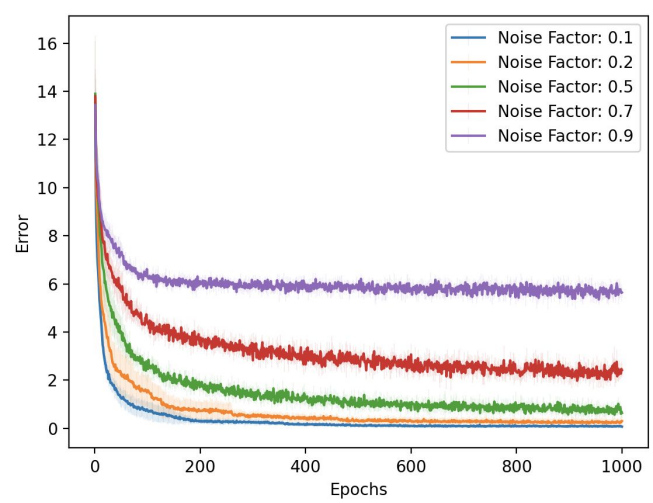
Parámetros:
n = 0.05
Epochs = 1000
Cantidad de ejecuciones = 5
F. de activación = Logística
Capas = [56]
Con adam
Momentum = 0.3

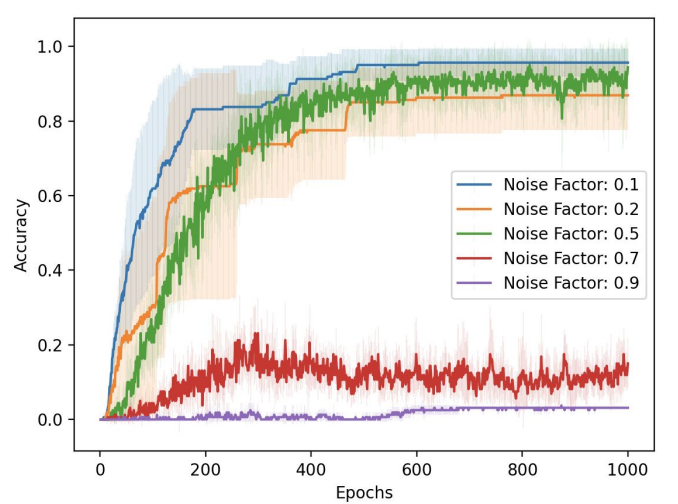




Eliminación de ruido (Salt)
Evolución de error y accuracy variando factor de ruido

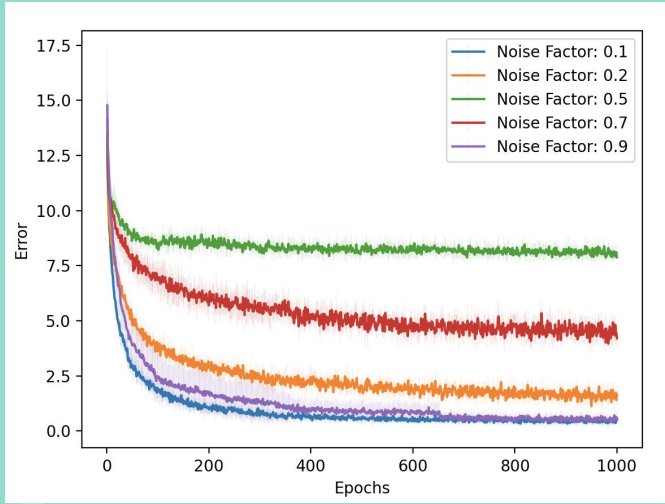
Parámetros:
n = 0.05
Epochs = 1000
Cantidad de ejecuciones = 5
F. de activación = Logística
Capas = [56]
Con adam
Momentum = 0.3

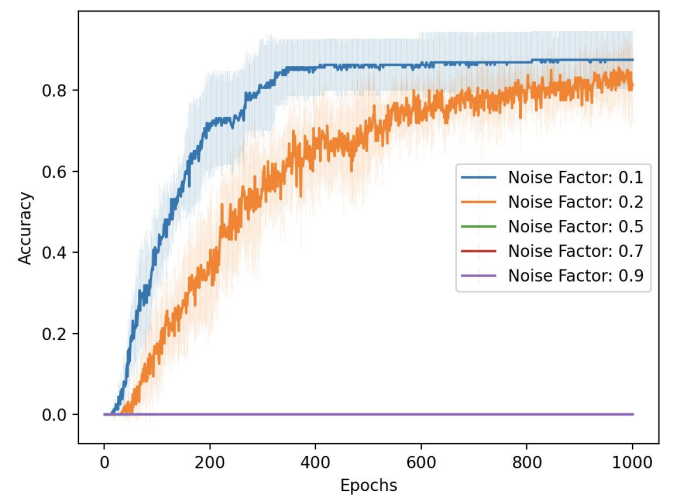




Eliminación de ruido (Salt & Pepper)
Evolución de error y accuracy variando factor de ruido

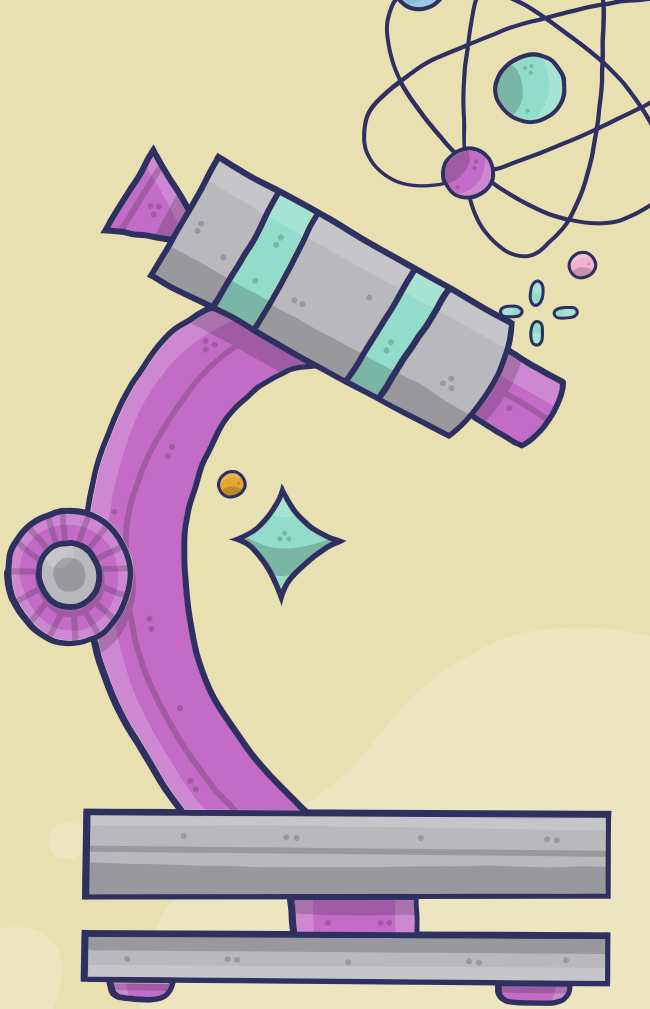
Parámetros:
n = 0.05
Epochs = 1000
Cantidad de ejecuciones = 5
F. de activación = Logística
Capas = [56]
Con adam
Momentum = 0.3





02.

Variable Autoencoder



Introducción VAE

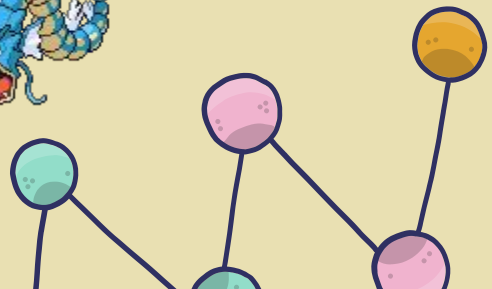
- A diferencia del encoder del AE, el VAE genera como salida hacia el espacio latente una distribución (varianza y media)
- Los VAE condicionan su salida a una distribución probabilística, por lo que cada uno de sus resultados tiene automáticamente un análisis directo (aunque no necesariamente correcto, eso depende de la calidad del modelo).
- La principal diferencia entre VAE y AE está en la pérdida calculada sobre la representación latente

2.1

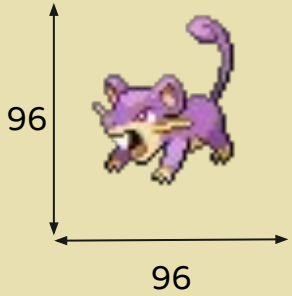
Dataset

Propiedades del dataset

- Se utilizarán como dataset distintos sprites de pokémon en formato png
- Las imágenes son de 96x96 a color y se encuentran sobre fondo transparente
- El dataset proviene de la primera generación de pokemones



Propiedades del dataset

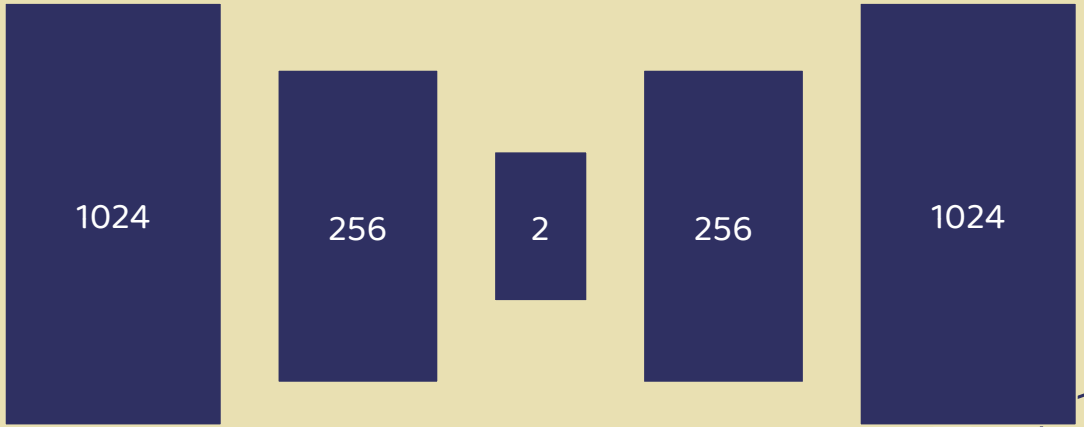


- En total, serán 150 sprites de 96x96 con 3 canales (RGB)
- Entrenado en una VAE utilizando *tensorflow-keras*.

2.2

Arquitectura

Arquitectura elegida



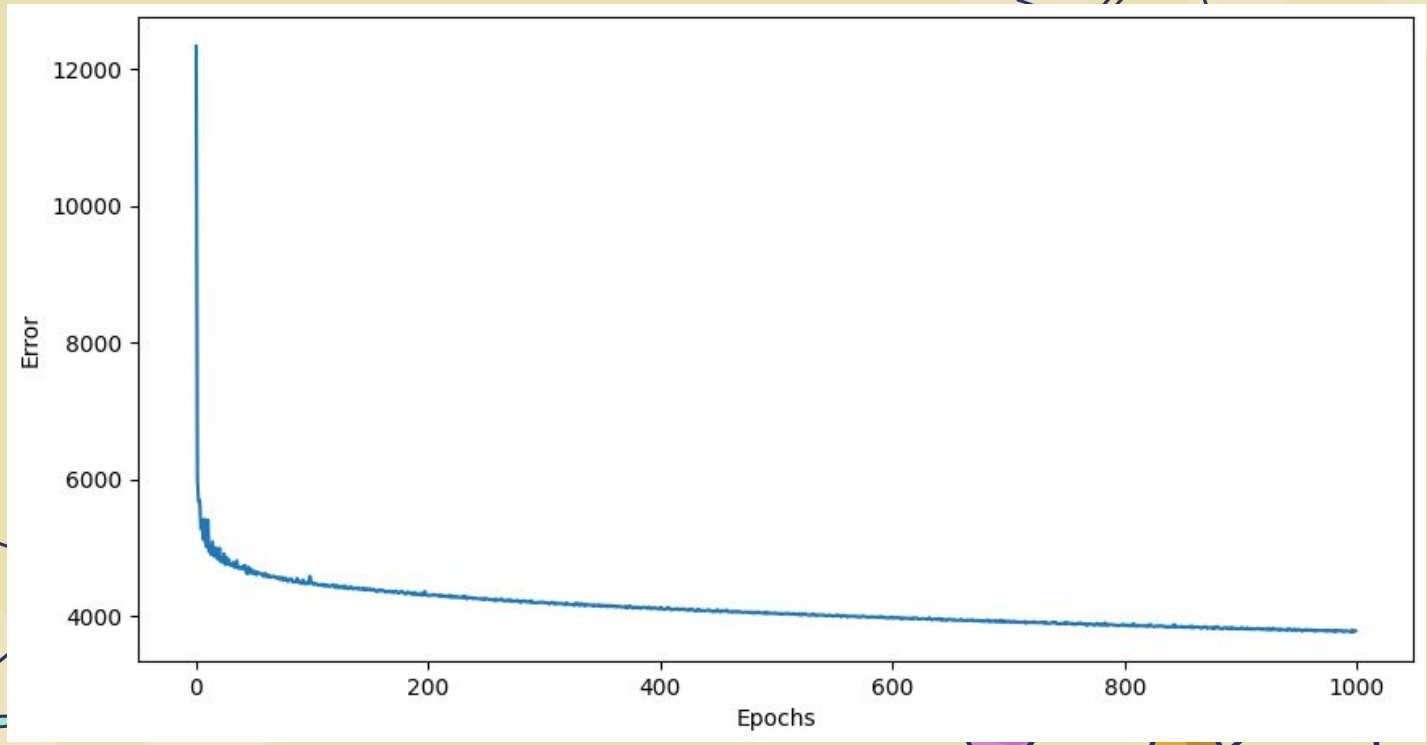
2.3 Resultados

Resultados
Evolución del error por época

Parámetros:

Epochs = 1000

Optimizador = RMSProp

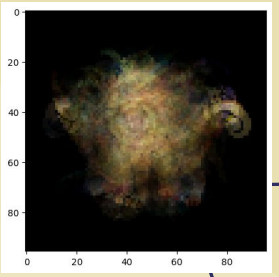


Resultados
Espacio latente

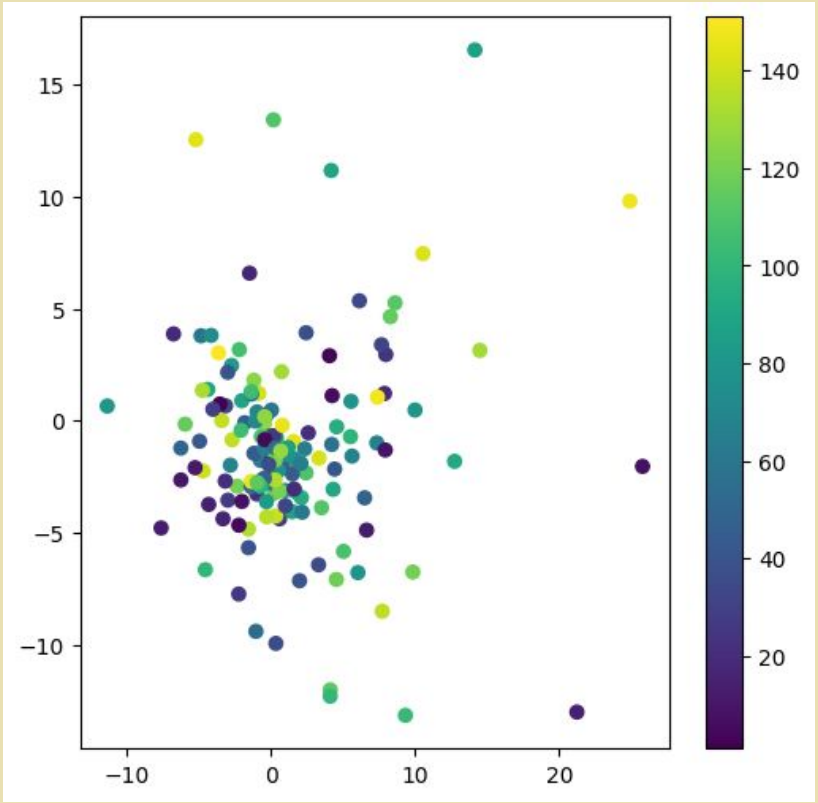
Parámetros:

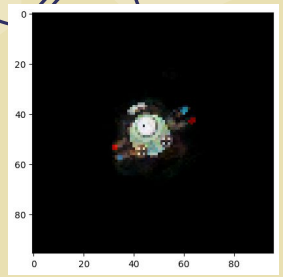
Epochs = 500

Optimizador = RMSProp

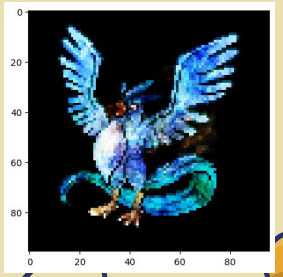


(0, 0)





(-10, 0.5)



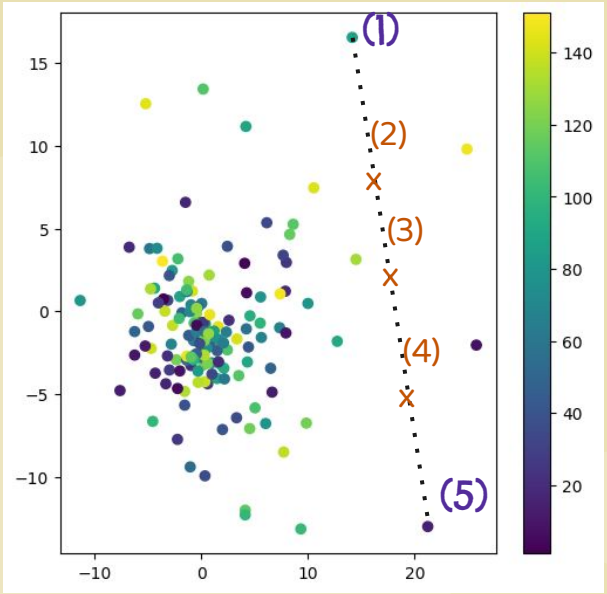
(-6, 14)

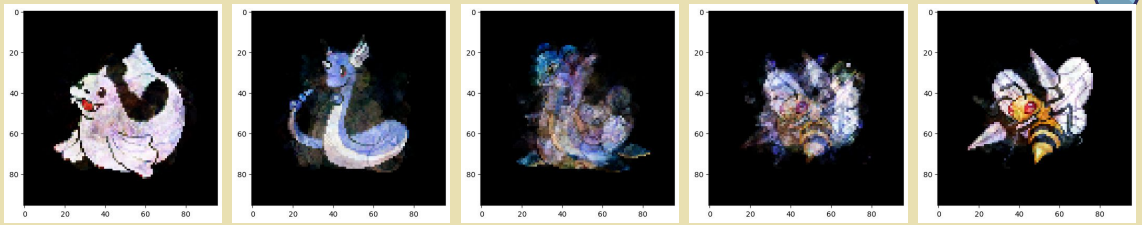
Resultados
Espacio latente

Parámetros:

Epochs = 500

Optimizador = RMSProp





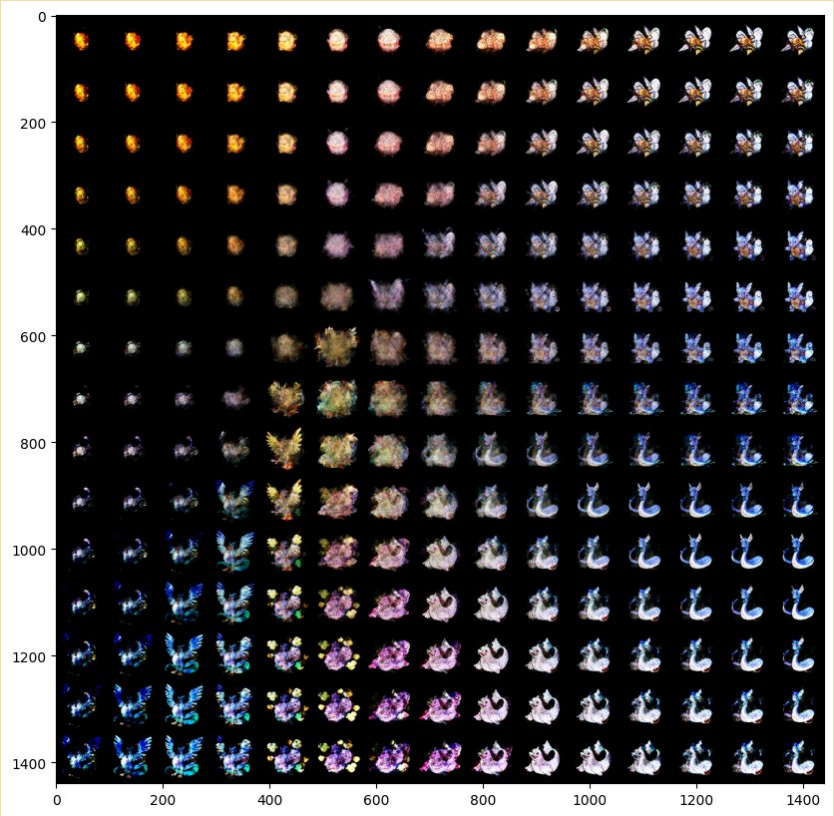
(1) (2) (3) (4) (5)

Resultados
Espacio latente

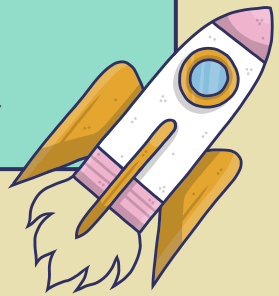
Parámetros:

Epochs = 500

Optimizador = RMSProp



¡DEMO!



04

Conclusiones

Conclusiones

- A menor dimensión del espacio latente, más difícil es recuperar la información que recibió el codificador.
- La selección de arquitectura se fundamenta empíricamente y dependiente del problema.
- Cuanto más impacto provoque el ruido, más difícil resulta el entrenamiento.
- Las imágenes generadas en zonas del espacio latente con densidad alta suelen ser borrosas.

Gracias!