

TP 4 - S.I.A - Grupo 1

Integrantes:

Burgos, Jose (61525)

Matilla, Juan Ignacio (60459)

Curti, Pedro (61616)

Panighini, Franco (61258)



Hopfield

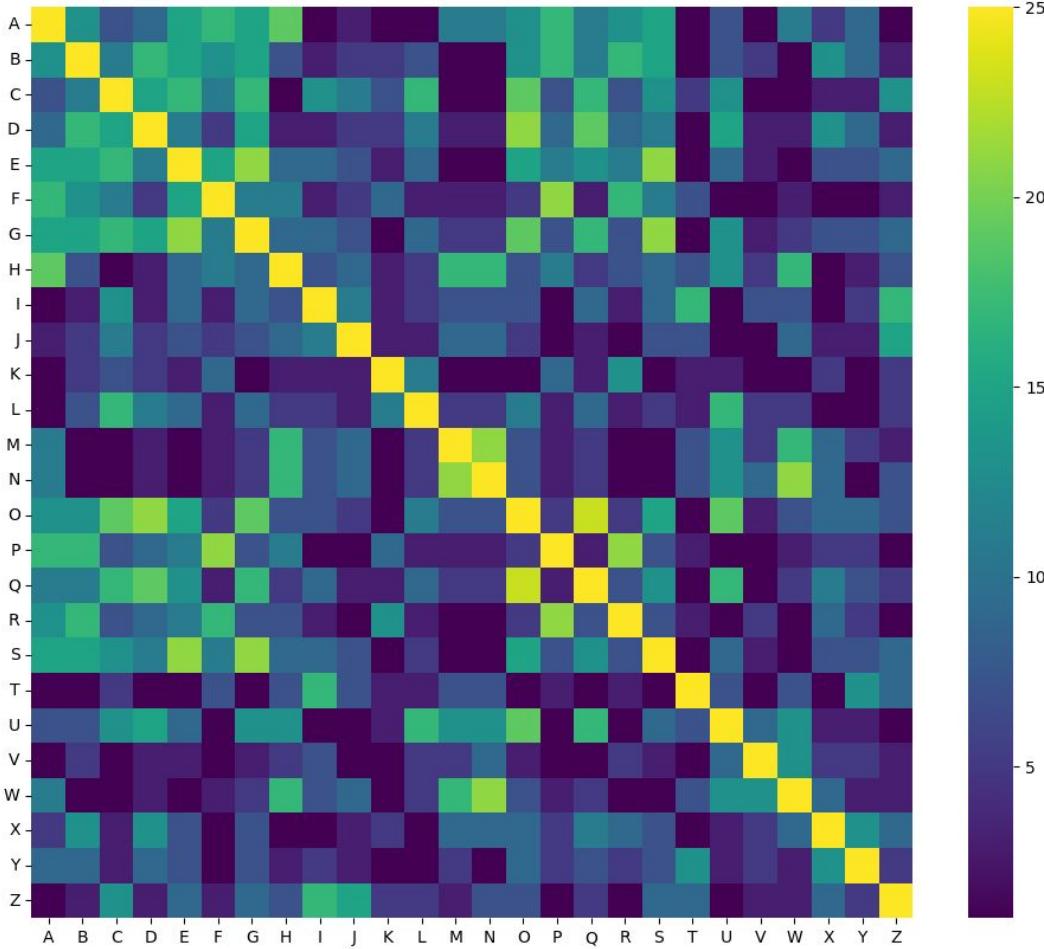
INPUT:
Matriz 5x5

Diccionario

- Selección de cantidad de letras mediante la regla del 15%:
 - $\# \text{Patrones almacenables} = 0.15 * \# \text{Neuronas} \approx 4$
- Selección de letras mediante el cálculo de ortogonalidad entre las mismas

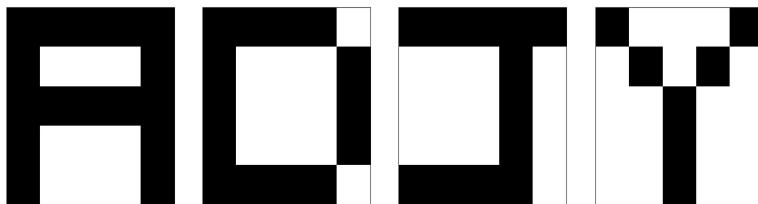
Diccionario - Idea

- Mayor valor → mayor similitud
 - Menor valor → menor similitud



Diccionario Ortogonal vs No Ortogonal

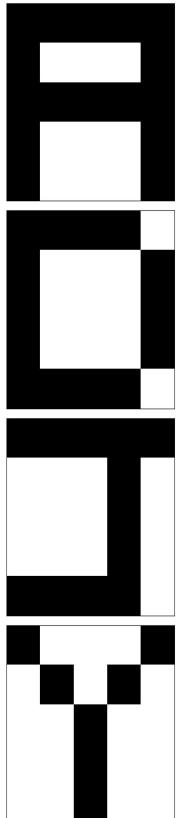
- Muy Ortogonal : A - D - J - Y:



- Poco Ortogonal: C- G- O- Q:



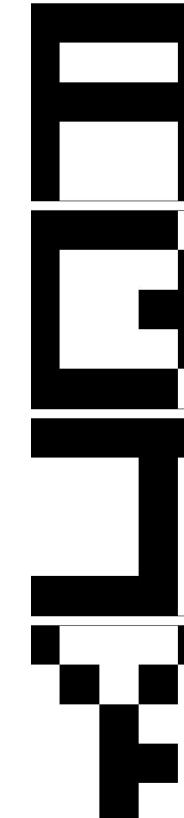
Funcionamiento



agregamos ruido
y alimentamos a la red



Vemos lo que recupero
la red



Correcta



Espuria

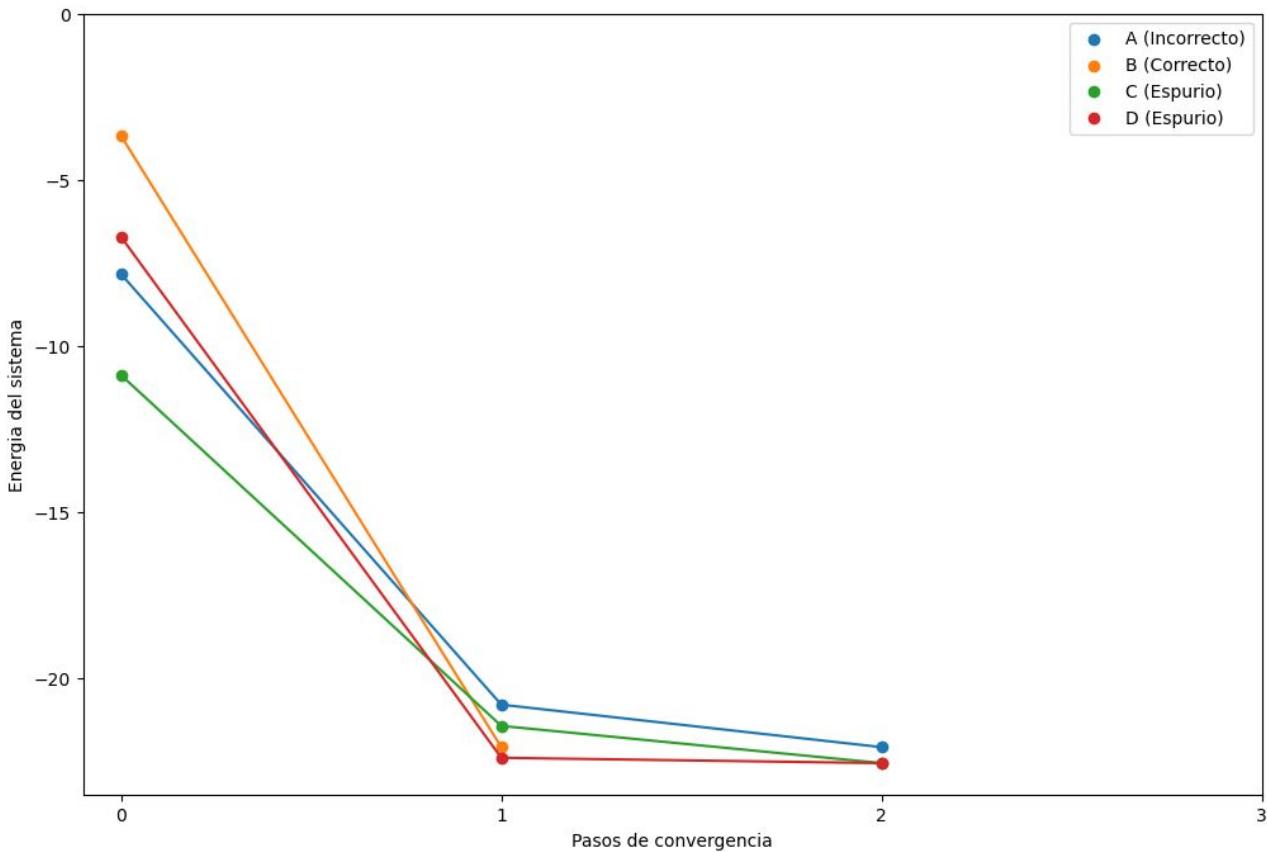


Correcta



Espuria

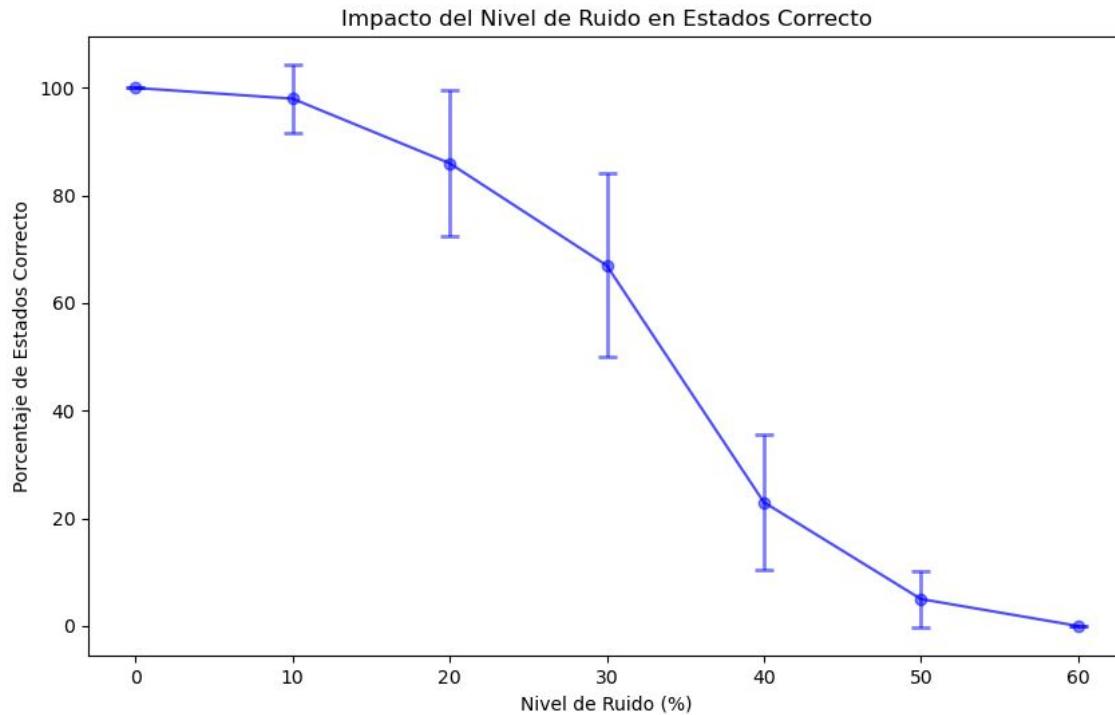
Energía



- Noise (α) : 0.2
- Min orthogonality: 1
- Max orthogonality: 20
- Mean Orthogonality: 15.25 ± 6.36

Ruido

Correctas



Runs: 10

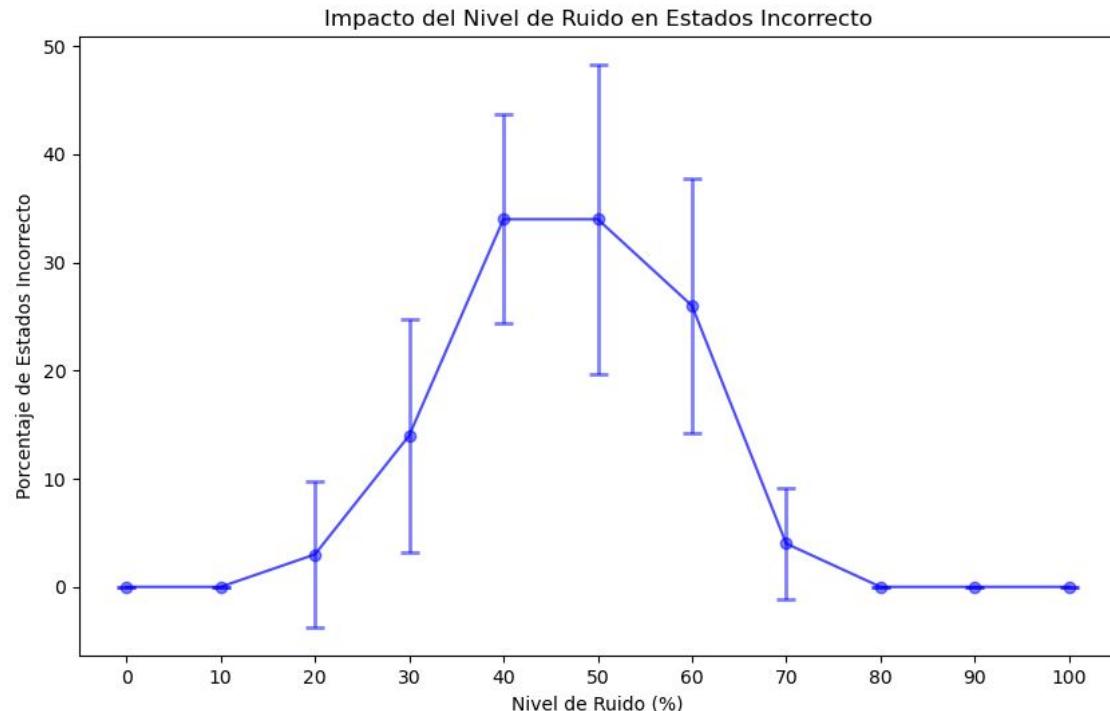
Letras: A, C, K, T

Ortogonalidad mínima: 0

Ortogonalidad máxima: 10

Promedio ortogonalidad:
3.67 +- 2.98

Incorrectas



Runs: 10

Letras: A, C, K, T

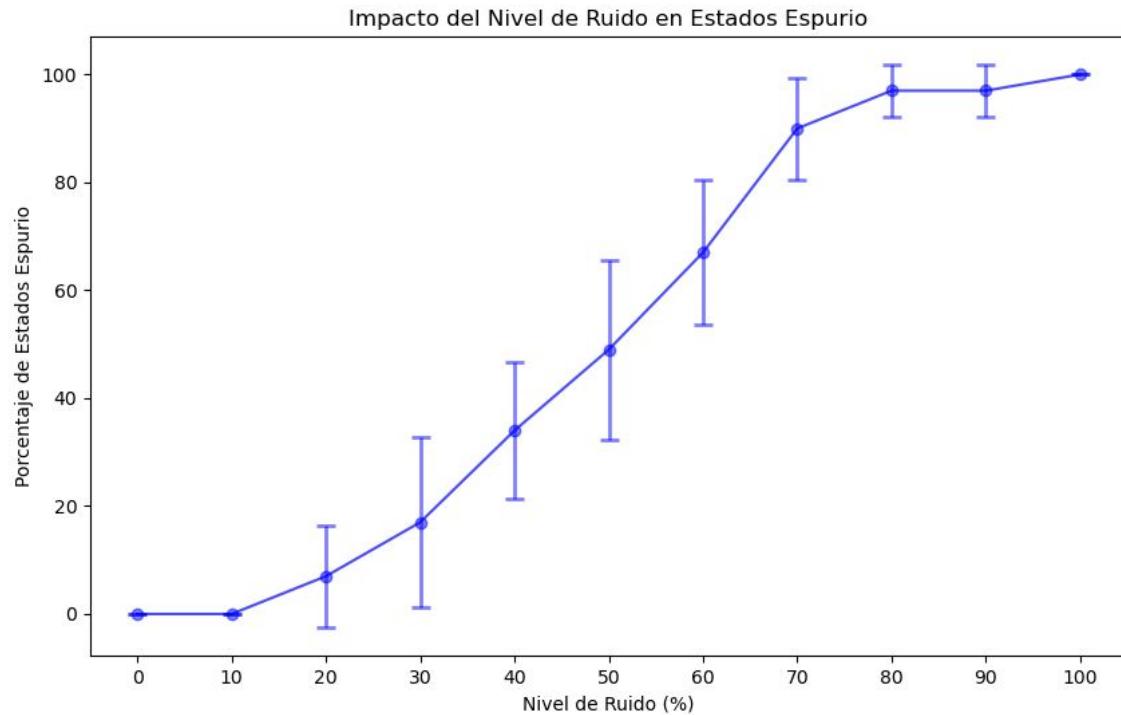
Ortogonalidad mínima: 0

Ortogonalidad máxima: 10

Promedio ortogonalidad:

3.67 +- 2.98

Espurias



Runs: 10

Letras: A, C, K, T

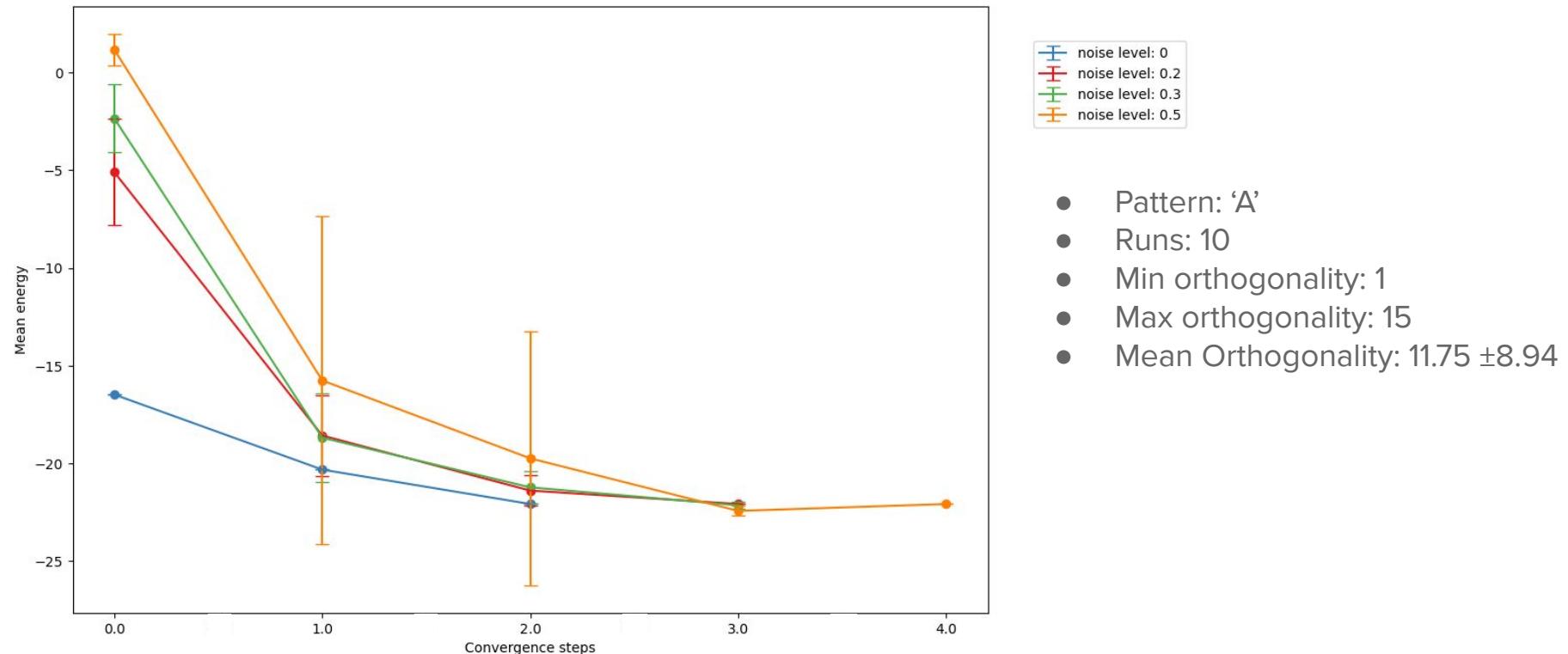
Ortogonalidad mínima: 0

Ortogonalidad máxima: 10

Promedio ortogonalidad:

3.67 +- 2.98

Energía

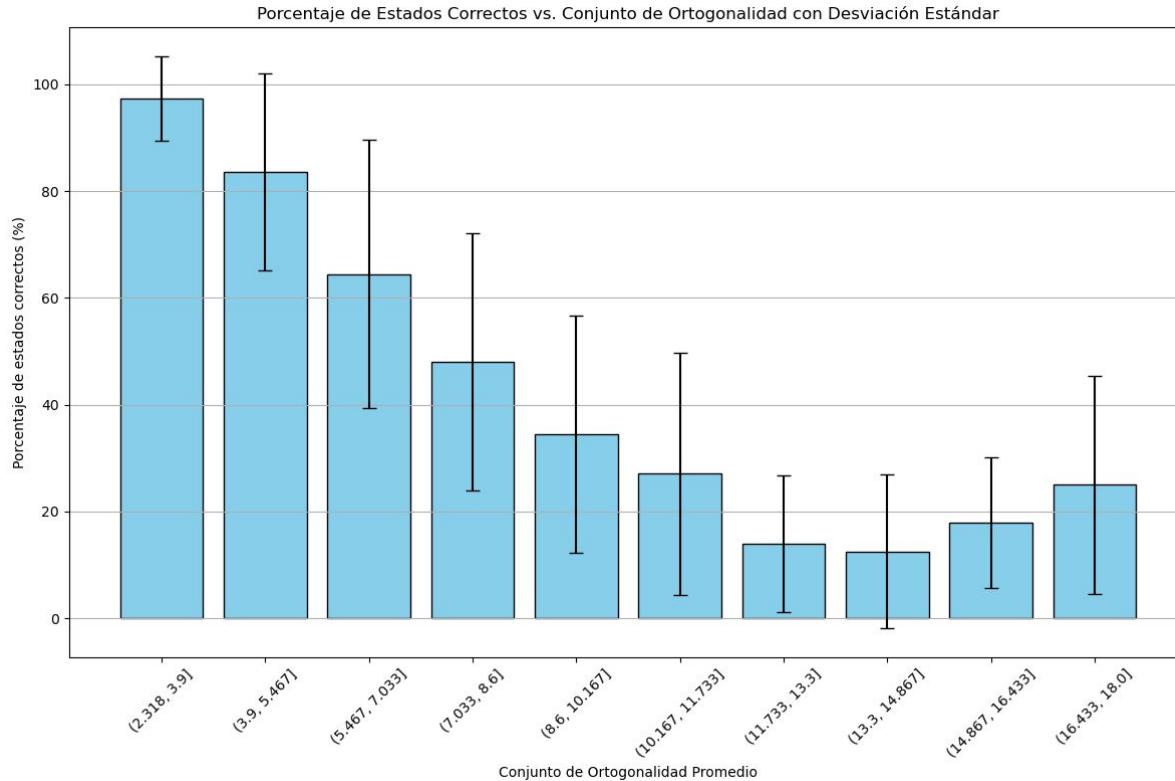


Conclusiones

- Mayor porcentaje de ruido:
 - Menor porcentaje de aciertos.
 - Mayor porcentaje de estados espurios.
 - Los estados incorrectos suben a un nivel de ruido intermedio pero bajan debido a la aparición de mayor estados espurios a niveles más elevados de ruido.

Ortogonalidad

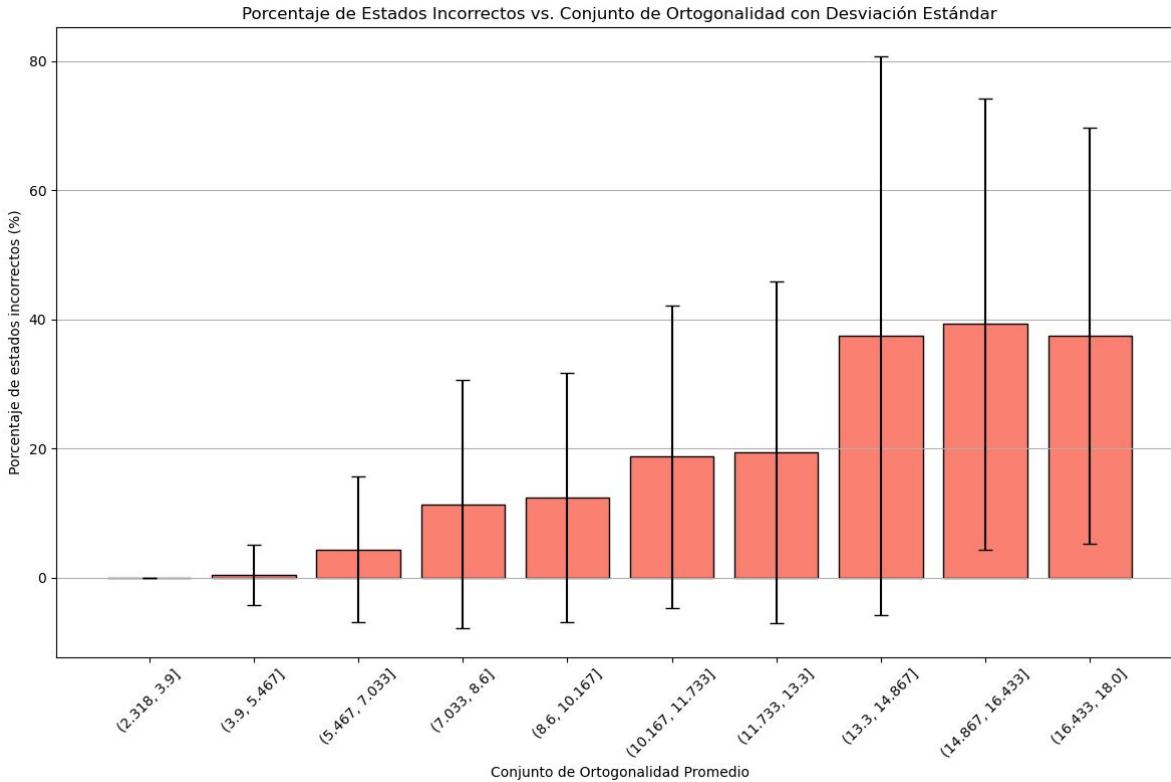
Correctas



Tamaño de la muestra:
Cantidad de conjuntos de 4
letras que cumplen la
condición de ortogonalidad
promedio y desvío estándar
respectiva, con máximo de
500 conjuntos

Ruido para pruebas: 0.1

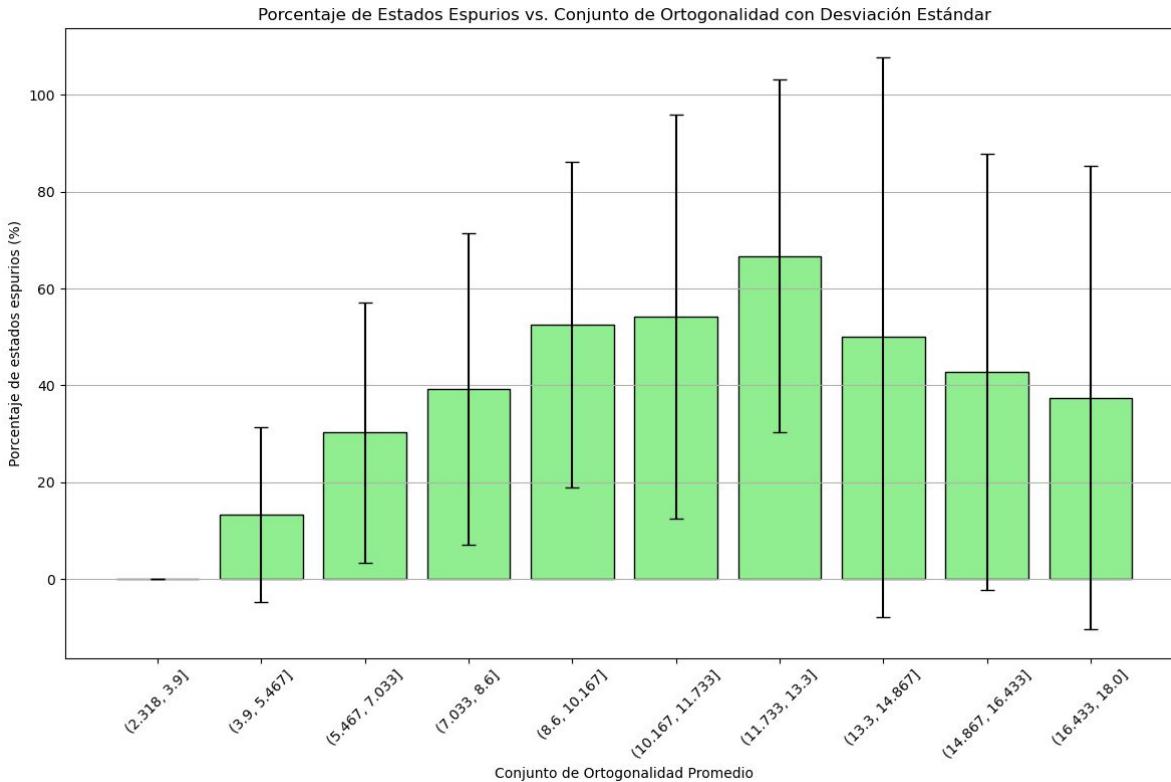
Incorrectas



Tamaño de la muestra:
Cantidad de conjuntos de 4
letras que cumplen la
condición de ortogonalidad
promedio y desvío estándar
respectiva, con máximo de
500 conjuntos

Ruido para pruebas: 0.1

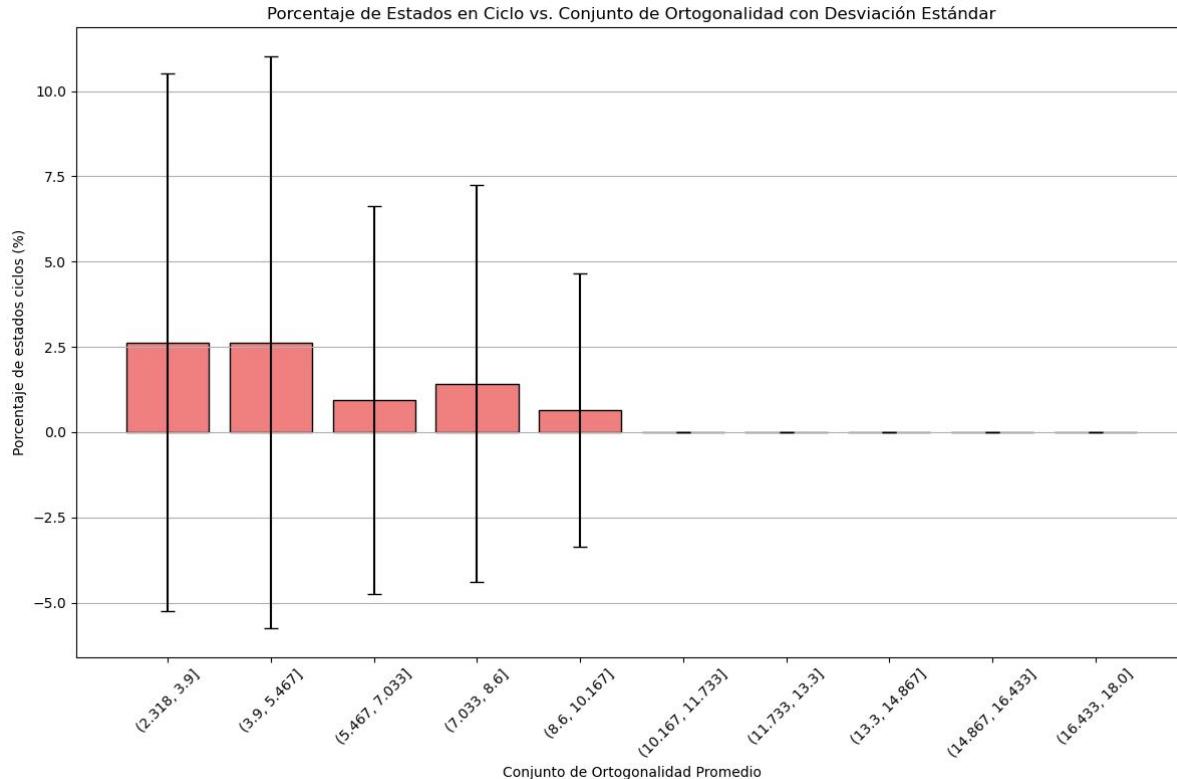
Espurias



Tamaño de la muestra:
Cantidad de conjuntos de 4
letras que cumplen la
condición de ortogonalidad
promedio y desvío estándar
respectiva, con máximo de
500 conjuntos

Ruido para pruebas: 0.1

Ciclos



Tamaño de la muestra:
Cantidad de conjuntos de 4
letras que cumplen la
condición de ortogonalidad
promedio y desvío estándar
respectiva, con máximo de
500 conjuntos

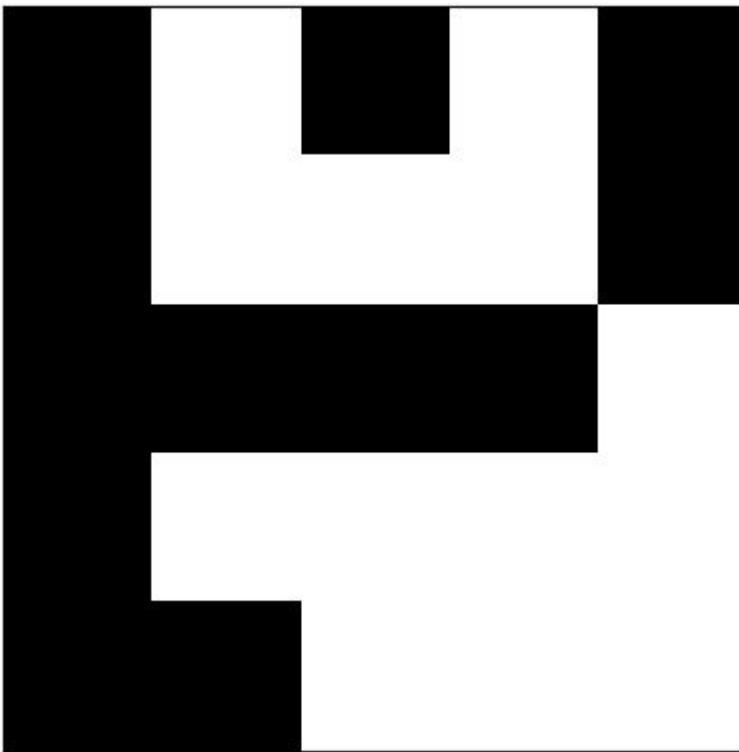
Ruido para pruebas: 0.1

Conclusiones

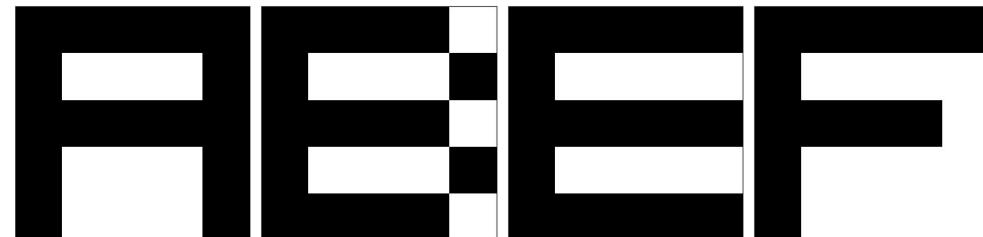
- Mayor nivel de ortogonalidad:
 - Mayor cantidad de aciertos .
 - Una pequeña chance de entrar en un estado ciclo.
- Menor nivel ortogonalidad:
 - Mayor cantidad de estados incorrectos
 - Mayor cantidad de espurios.

Ciclo

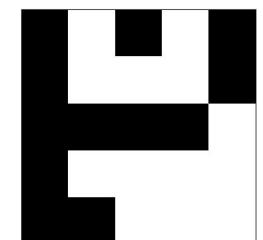
Ciclo



- Fed Patterns:



- Pattern to recover:



Conclusiones

- Los ciclos son más comunes en niveles de ortogonalidad superiores, puesto que a medida que desciende la ortogonalidad, aumentan los patrones espurios, donde se estanca la red, encontrando un falso mínimo, y esto provoca que no entre en ciclos.

INPUT: MNIST Digits

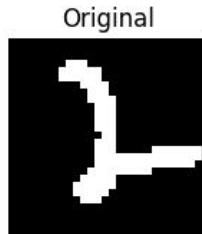
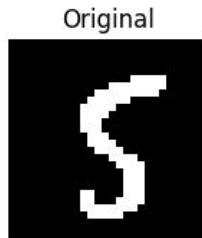
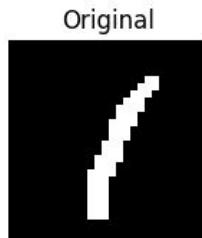
Tratamiento del input

- Transformar el dataset de la escala de grises a pixeles blancos y negros únicamente (Binarizar la matriz de cada input).
- Transformar los 0 en -1, para cumplir con el dominio de entrada de Hopfield: {-1, 1}.

```
# Cargar el dataset MNIST
mnist = fetch_openml(name='mnist_784', version=1, as_frame=False)
data = mnist['data']
targets = mnist['target'].astype(int)

# Binarizar los datos
patterns_pca = binarize(data, threshold=127).astype(int)
patterns_pca[patterns_pca == 0] = -1
```

Resultado - Todos estados espurios



Comparación de ortogonalidad

```
def calculate_average_orthogonality(patterns):
    # Normaliza los patrones para que cada vector tenga norma 1
    norms = np.linalg.norm(patterns, axis=1, keepdims=True)
    normalized_patterns = patterns / norms

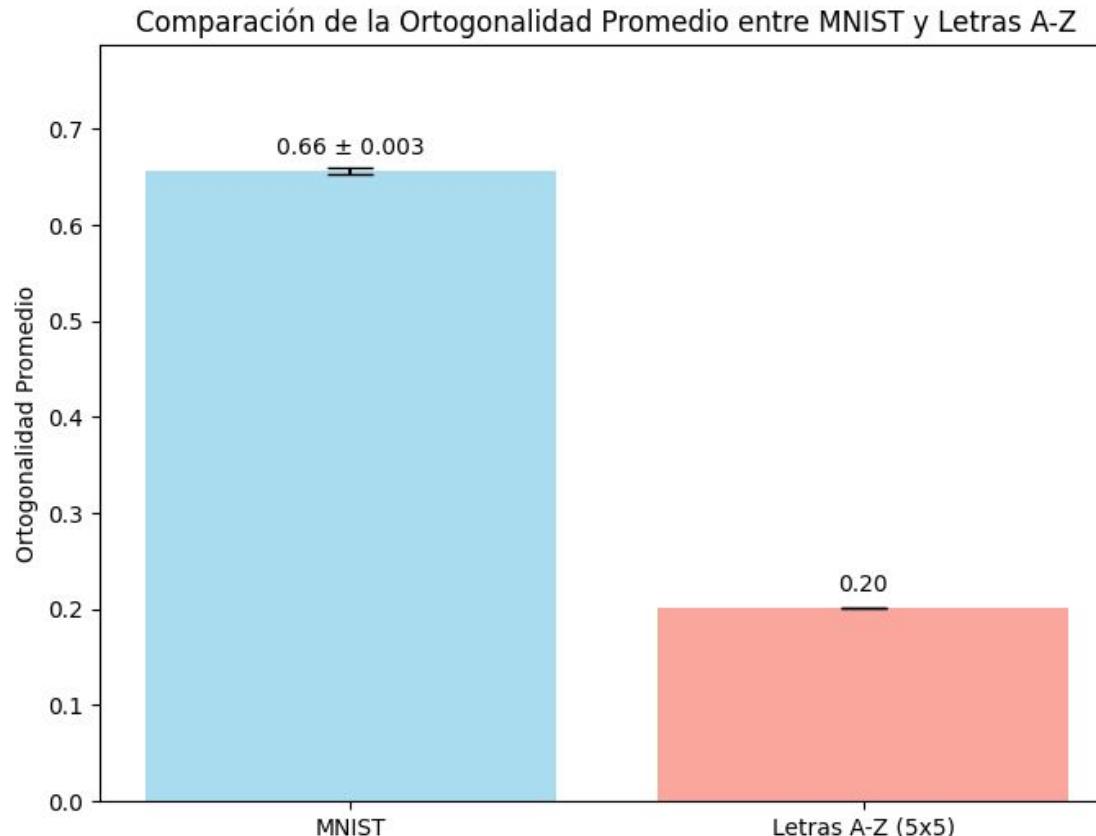
    K = normalized_patterns.shape[0]
    product_matrix = np.dot(normalized_patterns, normalized_patterns.T)

    triu_indices = np.triu_indices(K, k=1)
    upper_triangle = product_matrix[triu_indices]

    average_orthogonality = np.mean(upper_triangle)

    return average_orthogonality
```

Ortogonalidad de MNIST vs Matriz 5x5



Para MNIST:
1000 samples por run
10 runs

Conclusión

- La poca ortogonalidad de los patrones hace que sea imposible de reconocer por una red de Hopfield.
- Todos los inputs devuelven estados espurios, incluso tratando de almacenar pocos patrones.
- Debemos seguir otra estrategia.

**INPUT:
MNIST PCA**

Hipótesis

La imagen contiene mucha información innecesaria para representar el patrón que contiene

Si utilizamos PCA podemos reducir la dimensión y quedarnos con lo que realmente varía de input a input y así lograr que Hopfield logre reconocer los patrones

Tratamiento del input

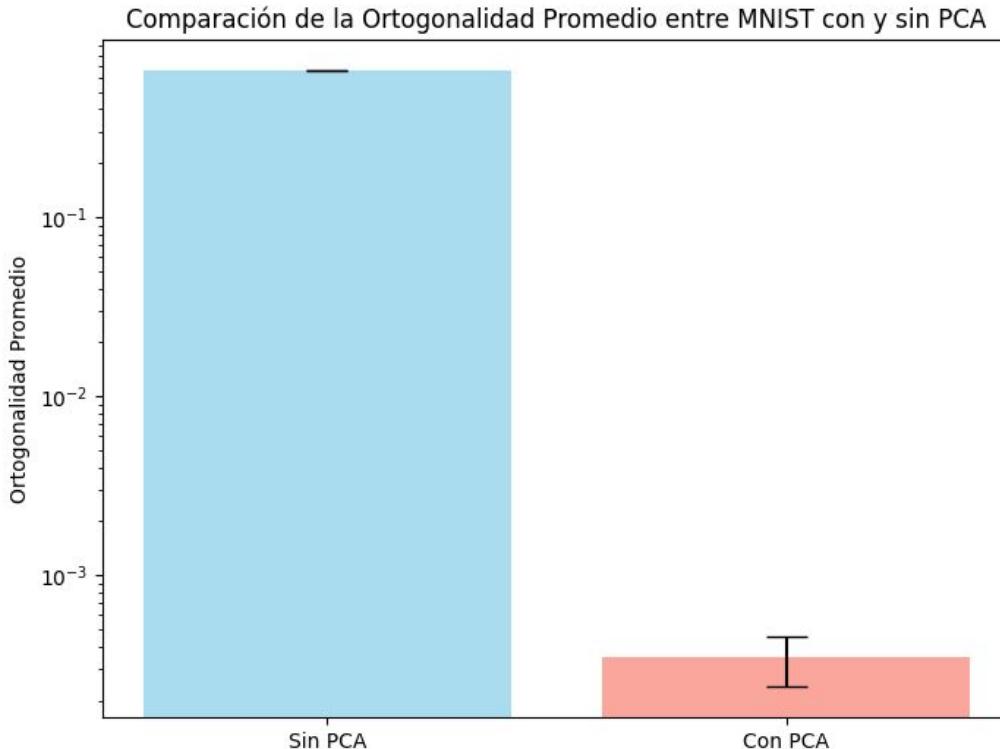
```
# Estandarizar los datos
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
pca_full = PCA().fit(data_scaled)

# Calcular cuantas componentes representan el 95% de la varianza
cumulative_variance = np.cumsum(pca_full.explained_variance_ratio_)
n_components = np.argmax(cumulative_variance >= 0.95) + 1

# Aplicar PCA
pca = PCA(n_components=n_components)
data_pca = pca.fit_transform(data_scaled)

# Binarizar las imágenes con umbral 0
patterns_pca = binarize(data_pca, threshold=0).astype(int)
patterns_pca[patterns_pca == 0] = -1
```

Resultado del tratamiento del input



Sin PCA:

- Media = 0.657,
- Desviación Estándar = 1.89e-03

Con PCA (332 componentes):

- Media = 3.53e-04,
- Desviación Estándar = 9.21e-05

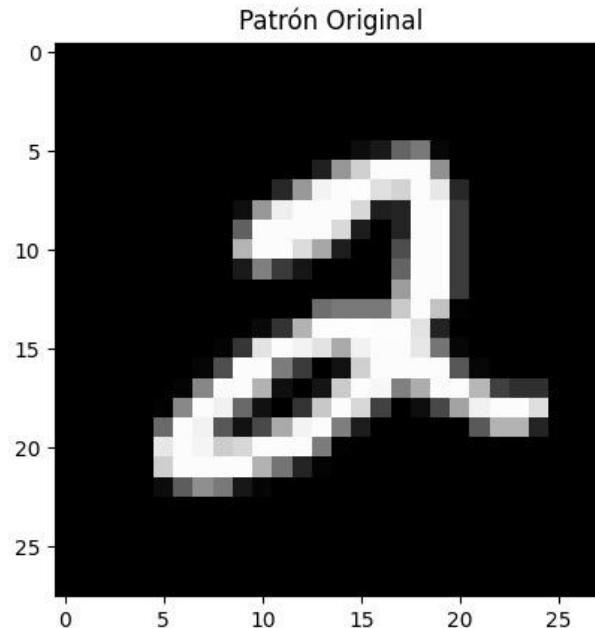
1000 samples por run

10 runs

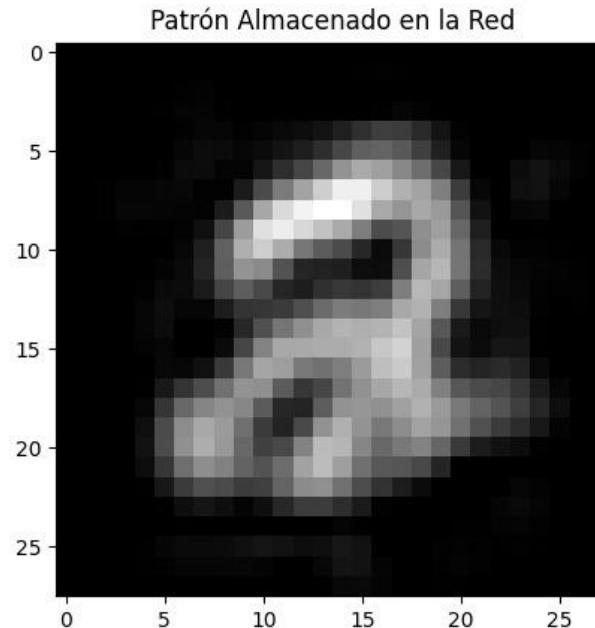
Revirtiendo PCA

```
# Recuperar el patrón
recovered_pattern, _, _ = hopfield_net.get_similar(test_pattern.copy(), max_iters: 100)
# Convertir el patrón recuperado a formato continuo
retrieved_pattern_continuous = recovered_pattern.astype(float)
# Invertir la transformación PCA
retrieved_pattern_original_space = pca.inverse_transform(retrieved_pattern_continuous)
# Desestandarizar el patrón reconstruido
retrieved_pattern_original_space_2d = retrieved_pattern_original_space.reshape(1, -1)
retrieved_pattern_descaled = scaler.inverse_transform(retrieved_pattern_original_space_2d)
retrieved_pattern_descaled = retrieved_pattern_descaled.flatten()
# Visualizar el patrón recuperado
plt.imshow(retrieved_pattern_descaled.reshape(28, 28), cmap='gray')
plt.title('Patrón Recuperado Reconstruido')
plt.show()
```

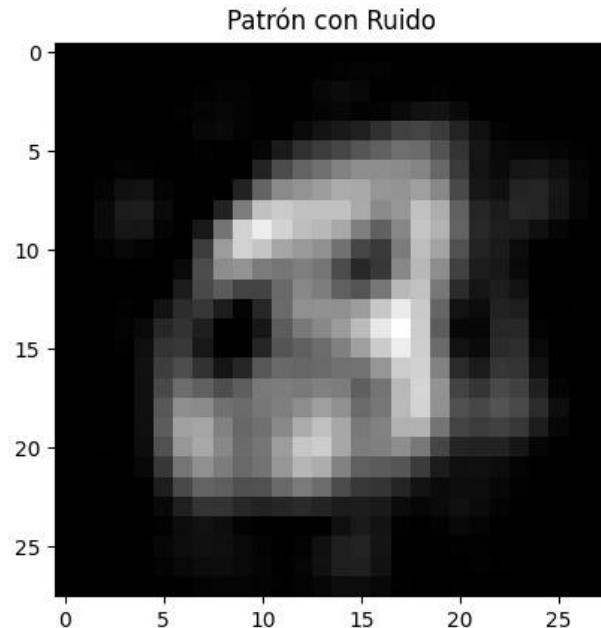
Ejemplo de resultado



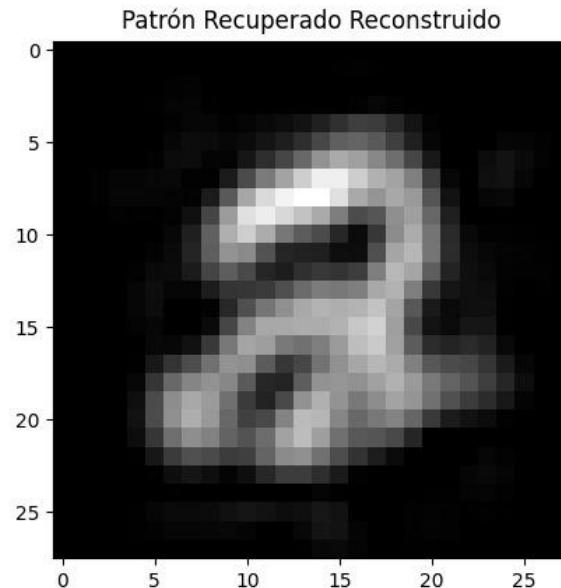
PCA



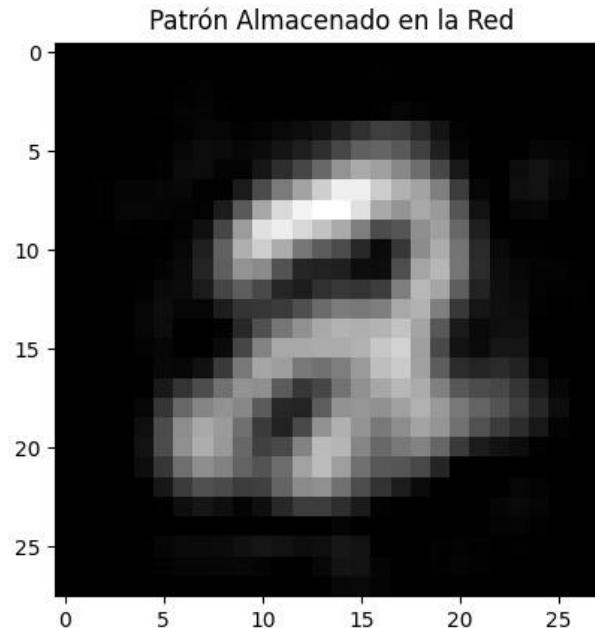
Ejemplo de resultado



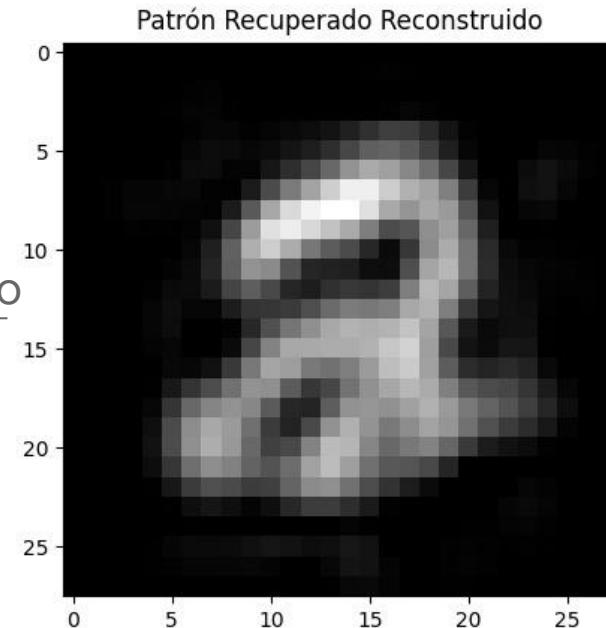
La red
recupera
el patrón



Ejemplo de resultado

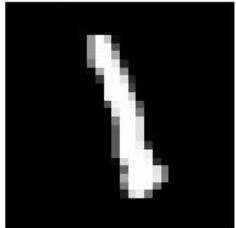


son el mismo

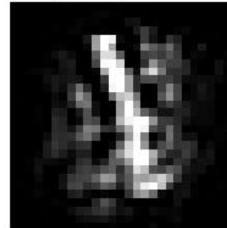


Más ejemplos

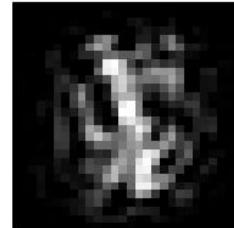
Original (Sin PCA)



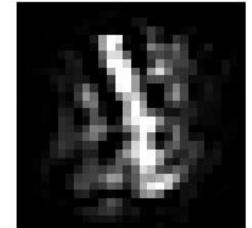
Original post PCA



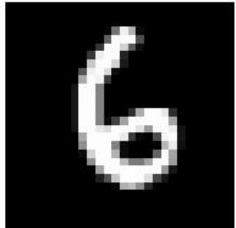
Con Ruido (20%)



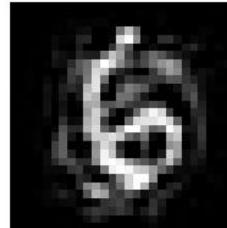
Recuperado



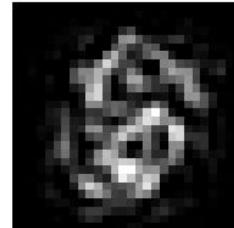
Original (Sin PCA)



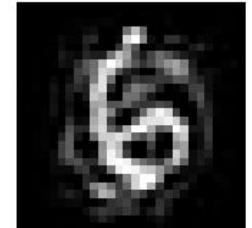
Original post PCA



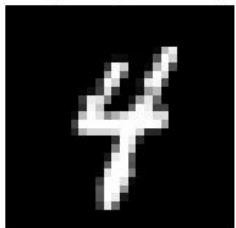
Con Ruido (20%)



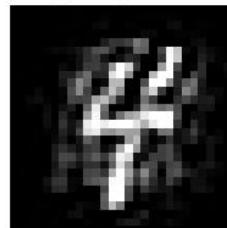
Recuperado



Original (Sin PCA)



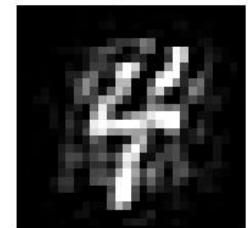
Original post PCA



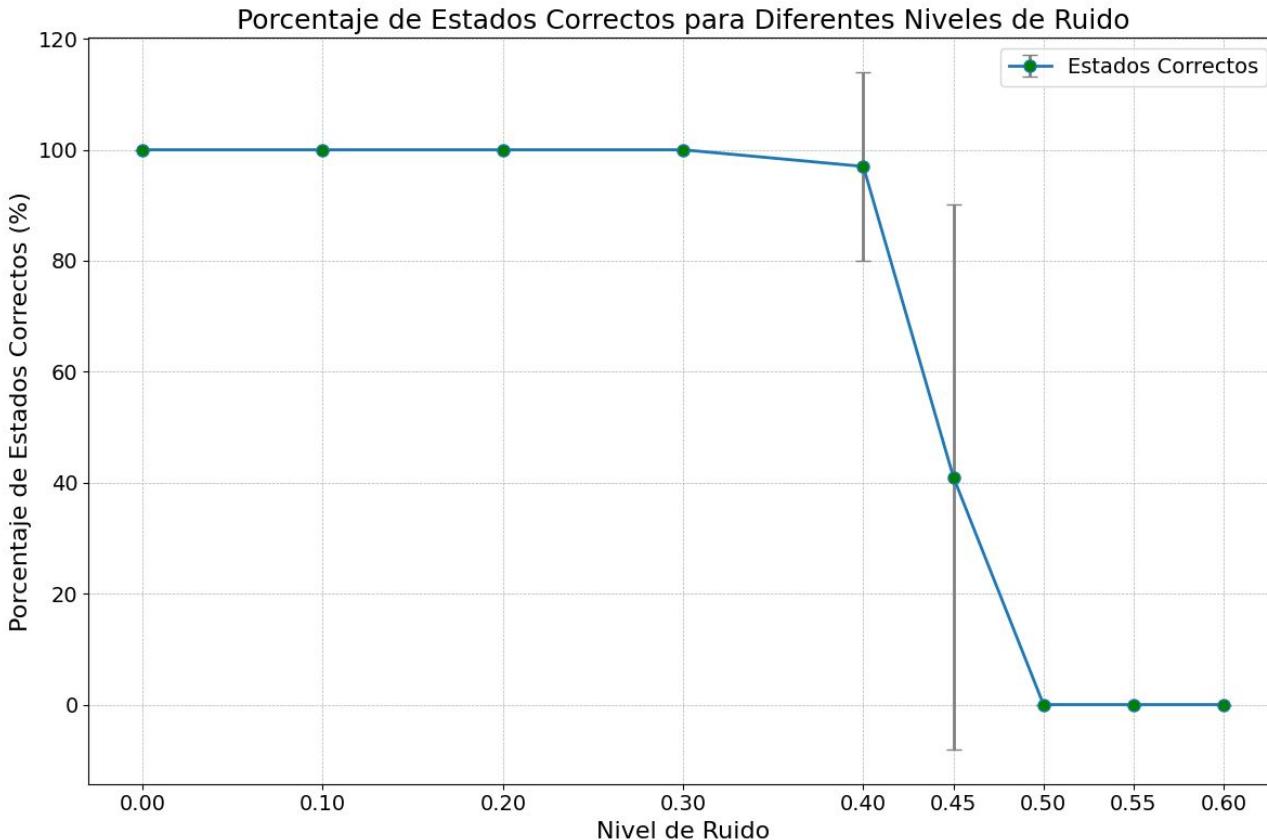
Con Ruido (20%)



Recuperado



Correctas



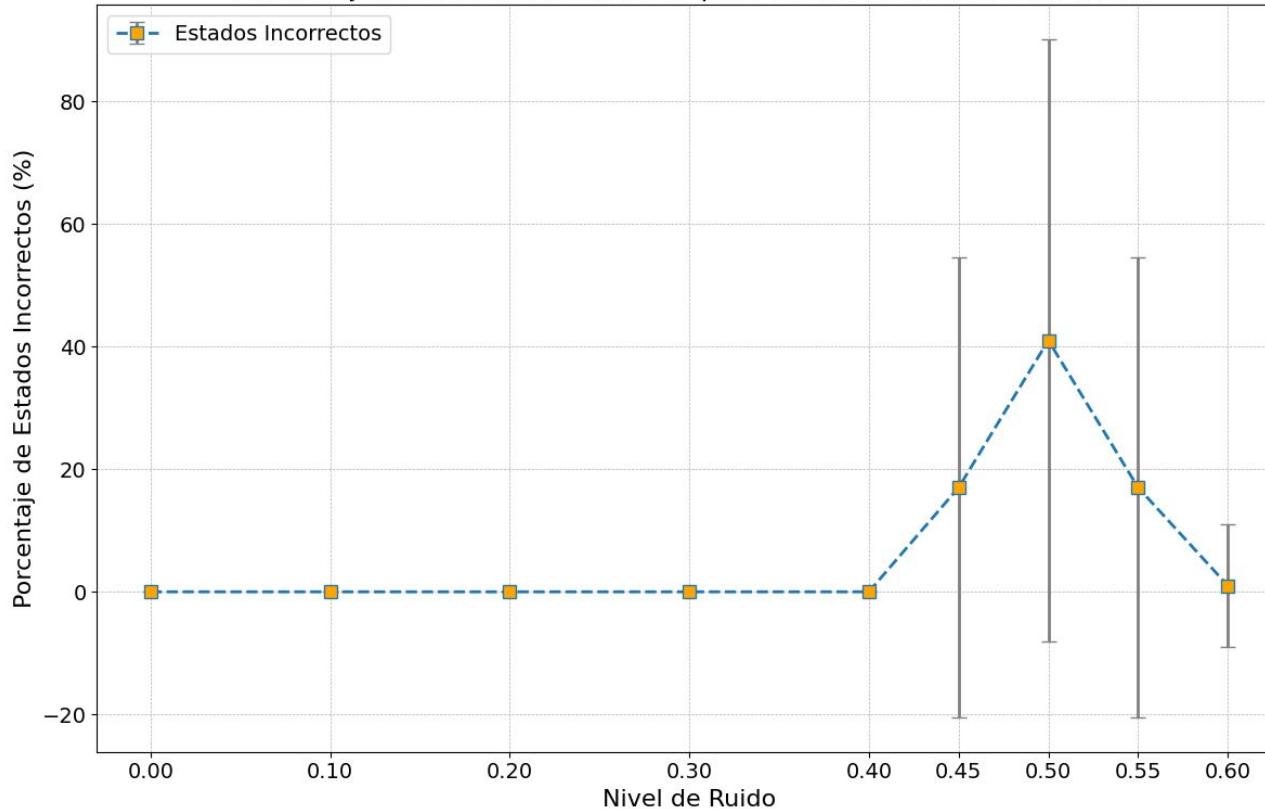
10 runs

Almacenando 1
patrón por dígito

Testeando cada
dígito

Incorrectas

Porcentaje de Estados Incorrectos para Diferentes Niveles de Ruido

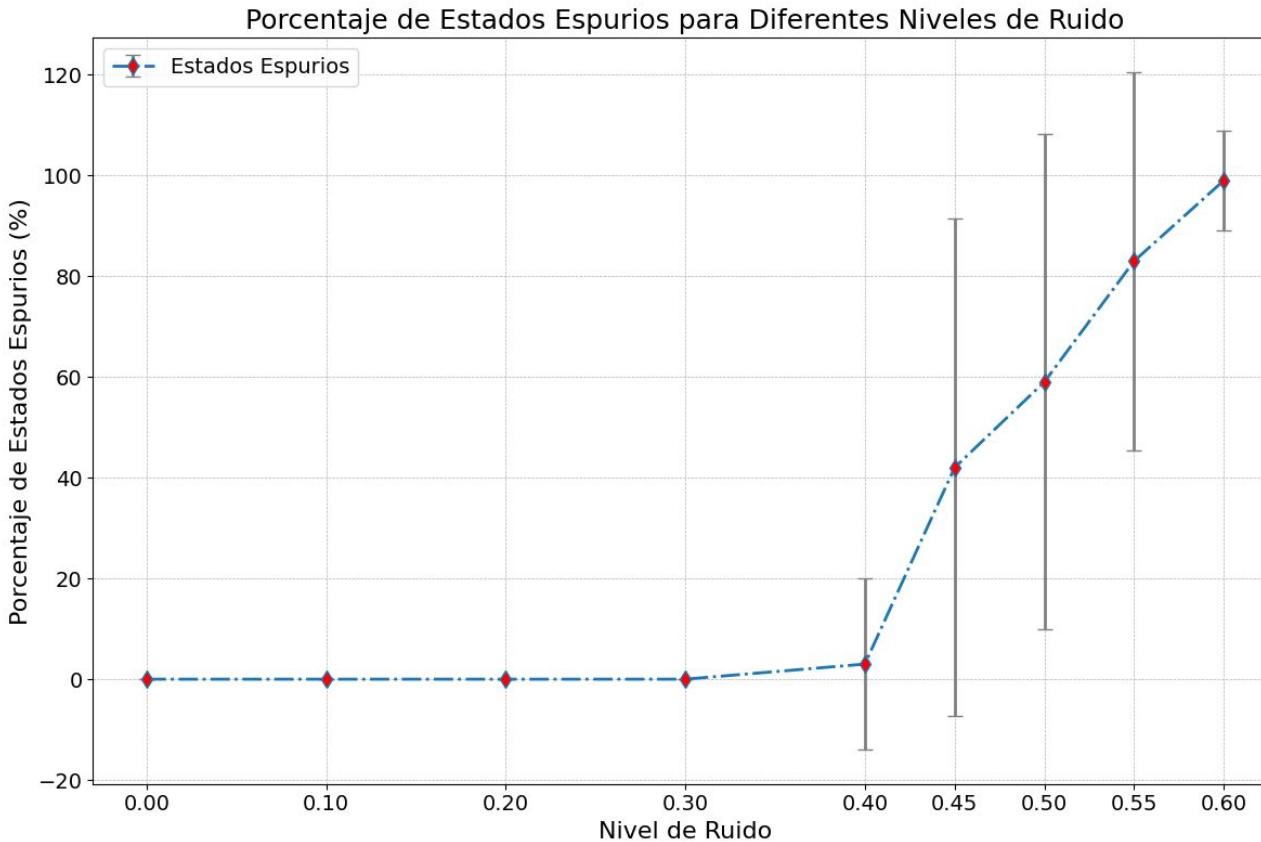


10 runs

Almacenando 1
patrón por dígito

Testeando cada
dígito

Espurios



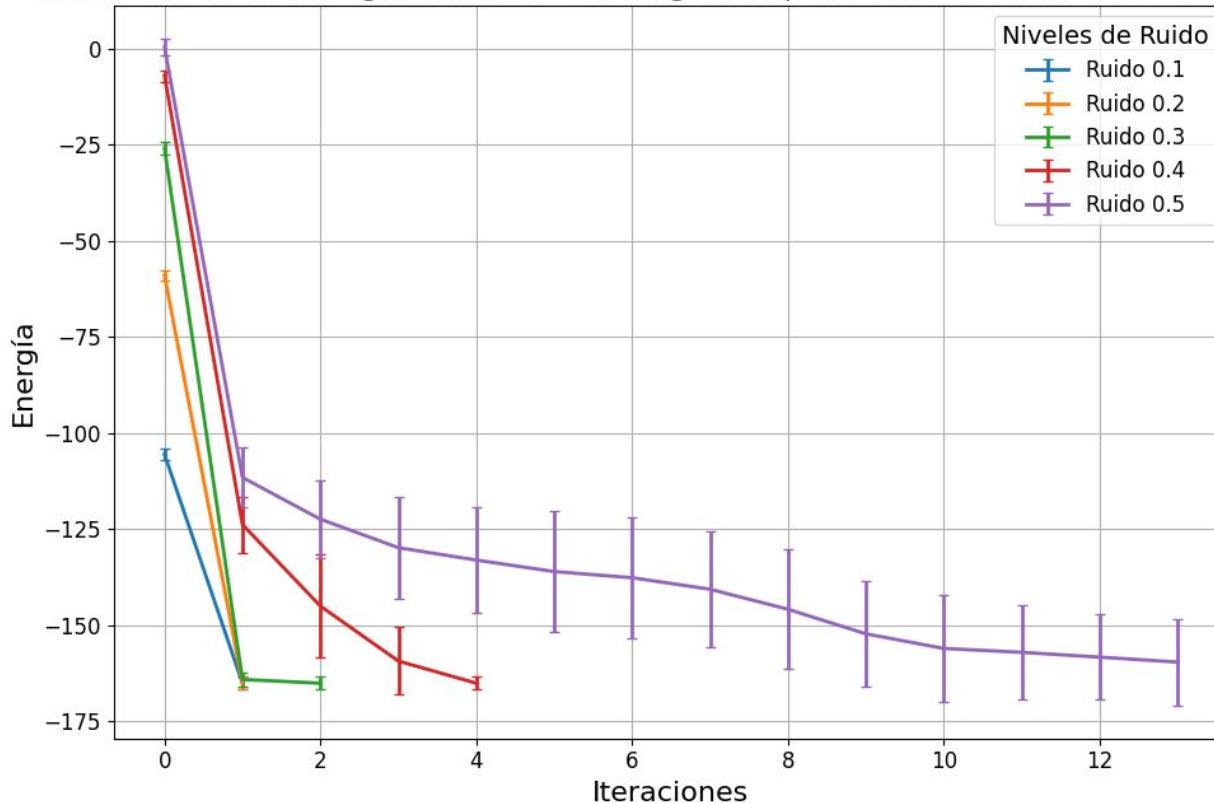
10 runs

Almacenando 1
patrón por dígito

Testeando cada
dígito

Energía hasta converger

Evolución de la Energía hasta la Convergencia para Diferentes Niveles de Ruido

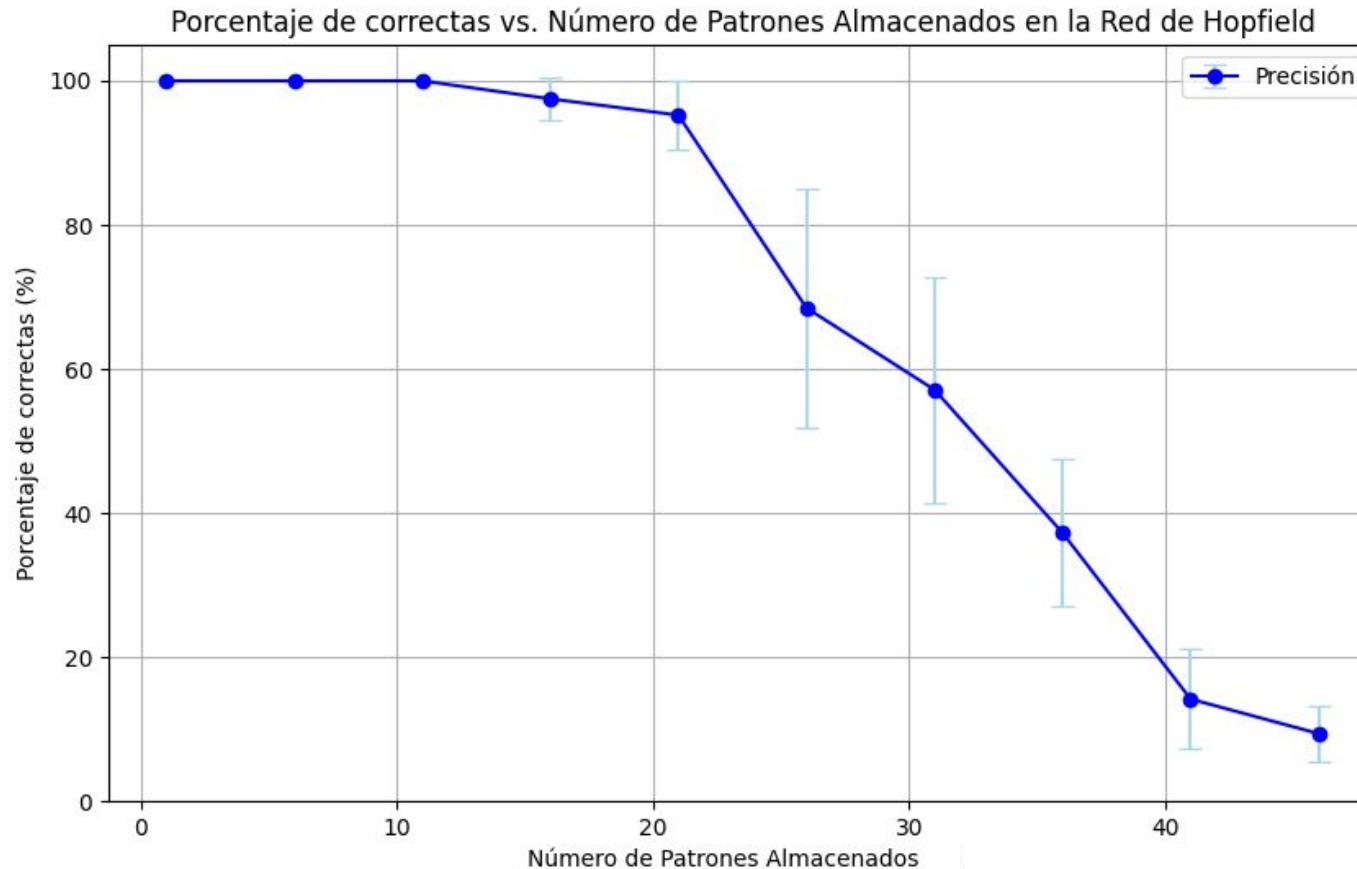


10 runs

Almacenando 1
patrón por
dígito

Testeando cada
dígito

Cantidad de patrones almacenados



10 runs

0.2 de noise

Testeando
para cada
patrón
almacenado

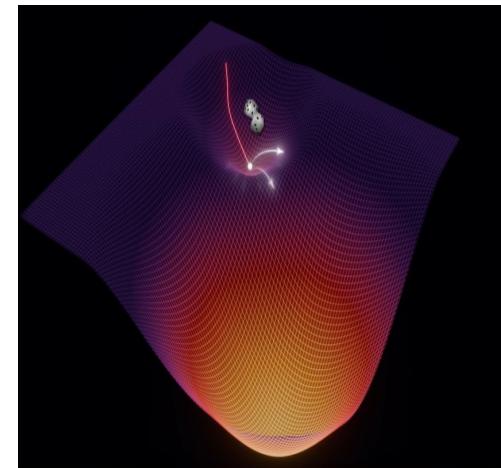
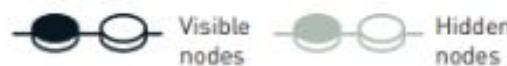
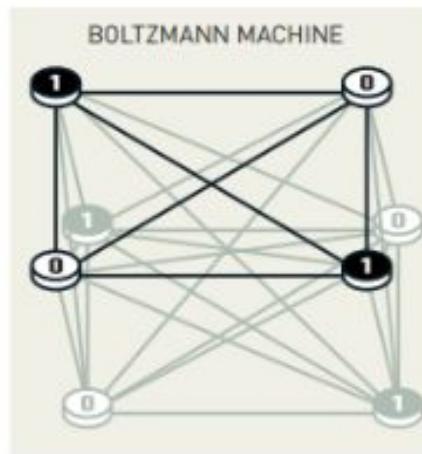
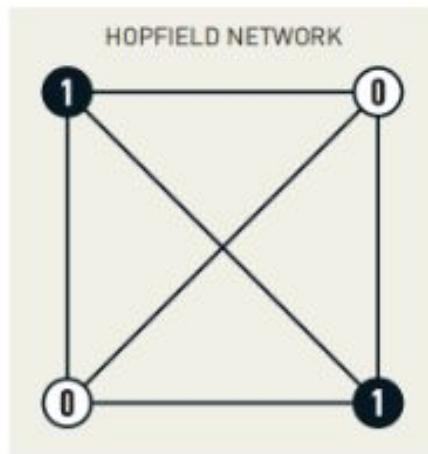
Conclusión

- Ventajas de utilizar PCA:
 - Al utilizar este preprocesamiento se logra poder utilizar el dataset de MNIST
 - Logra mayor resistencia al ruido, hasta el 40% para 1 patrón por dígito
- Desventajas o problemas a tener en cuenta:
 - La transformación a PCA provoca **pérdida de información**.

Restricted Boltzmann Machine (RBM)

Boltzmann machine vs Hopfield

Boltzmann agrega una componente estocástica a los pasos hacia la convergencia que logra escapar mínimos locales, agrega una capa de “hidden neurons” a la arquitectura de la red y agrega aprendizaje mediante contrastive divergence



Contrastive divergence

Positive phase

0 1 2 3 4
5 6 7 8 9

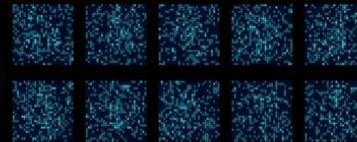
Collect pairwise activations
from training examples

$$\langle x_i x_j \rangle_{data}$$

Weight update

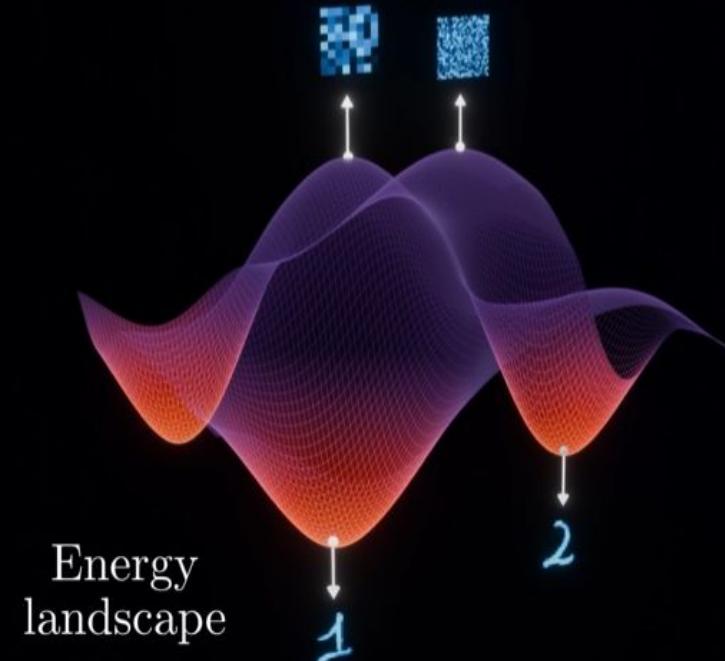
$$\Delta w_{ij} = \langle x_i x_j \rangle_{data} - \langle x_i x_j \rangle_{model}$$

Negative phase



Generate fictitious states
by running the update rule
and collect activations

$$\langle x_i x_j \rangle_{model}$$

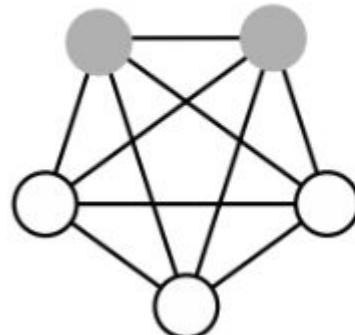


https://www.youtube.com/watch?v= bqa_l5hNAo&t=174s

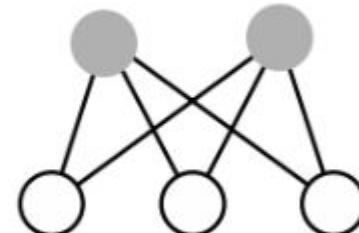
Restricted Boltzmann Machine vs Boltzmann Machine

Las RBM se diferencian de las BM debido a la ausencia de conexiones entre unidades visibles y ocultas haciendo más eficiente el cálculo de aprendizaje

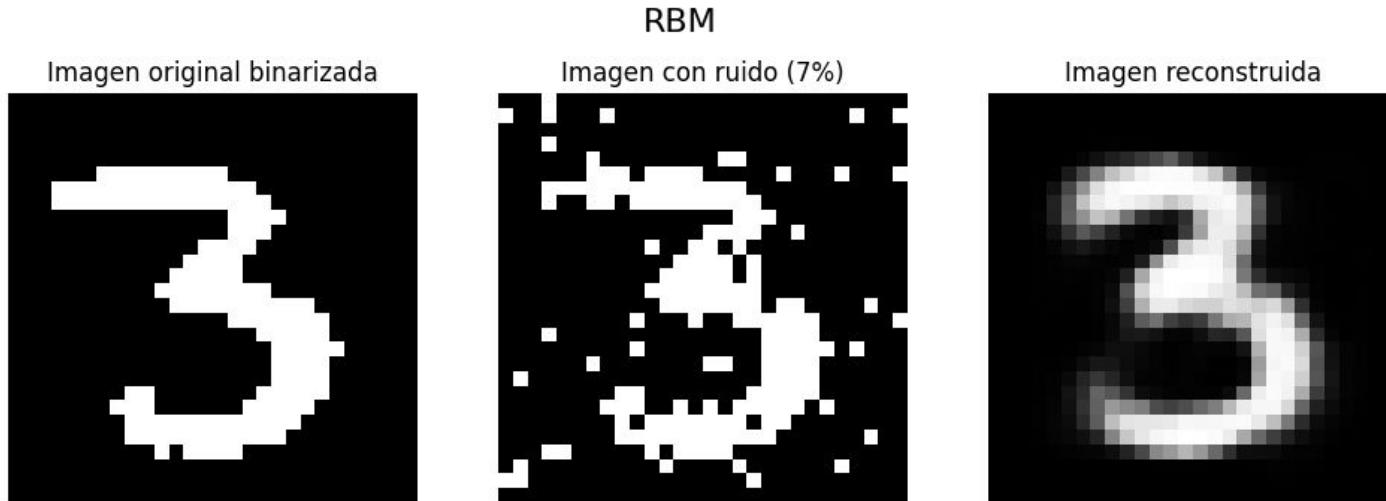
Boltzmann Machine



Restricted Boltzmann
Machine



Algunos resultados

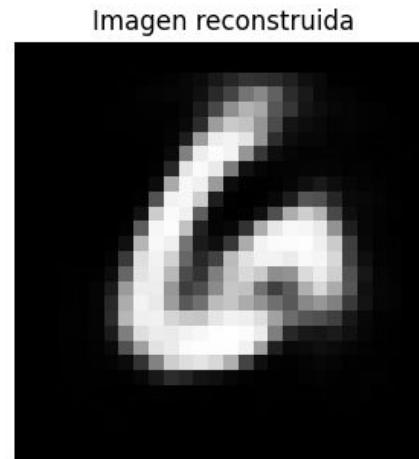
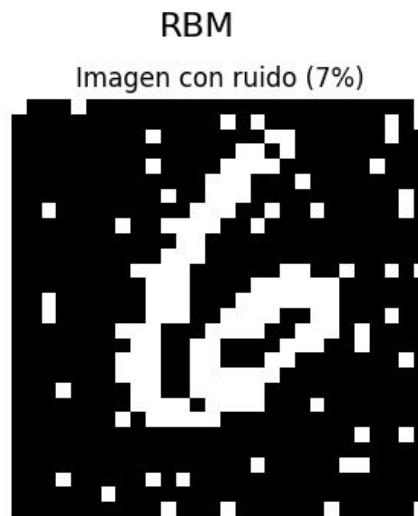
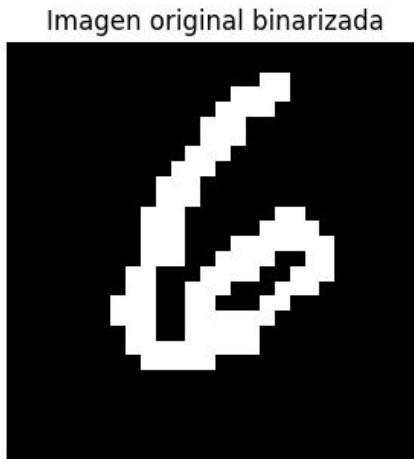


Estructura de red: **784 - 2048**

Learning rate: **0.01**

Cantidad de epochs: **5**

Algunos resultados



Estructura de red: **784 - 2048**

Learning rate: **0.01**

Cantidad de epochs: **5**

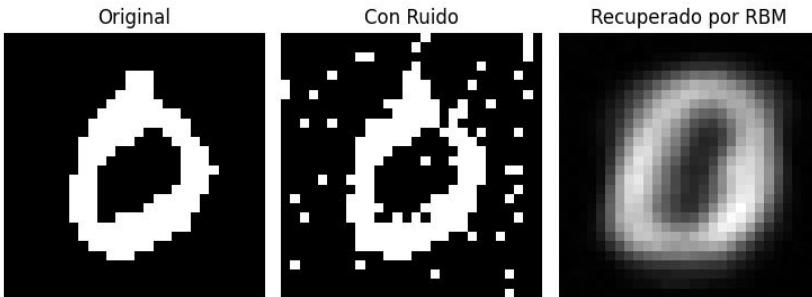
Hiperparametros

SSIM (Structural Similarity Index Measure)

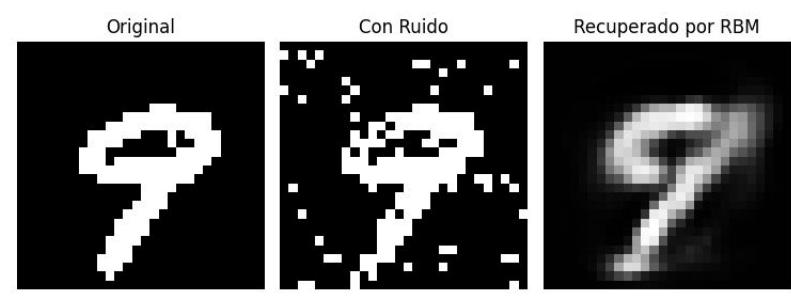
- Evalúa cambios en la **estructura**, luminancia y contraste de las imágenes.
- Es mejor métrica para evaluar patrones que una comparación pixel a pixel
- Varía entre -1 y 1, donde:
 - 1: Las imágenes son estructuralmente idénticas.
 - 0: No hay similitud estructural.
 - Valores negativos: Las imágenes son completamente diferentes.

SSIM - Ejemplos entre original y recuperado

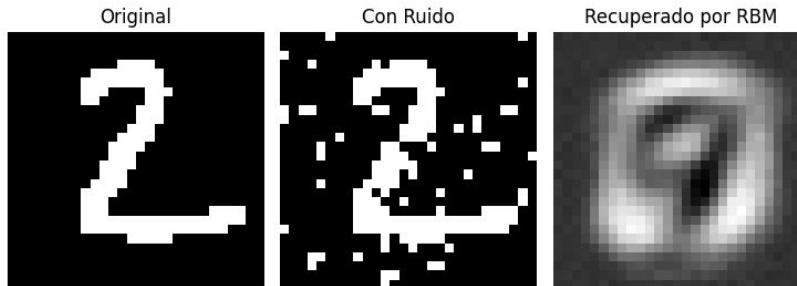
SSIM: 0.2827



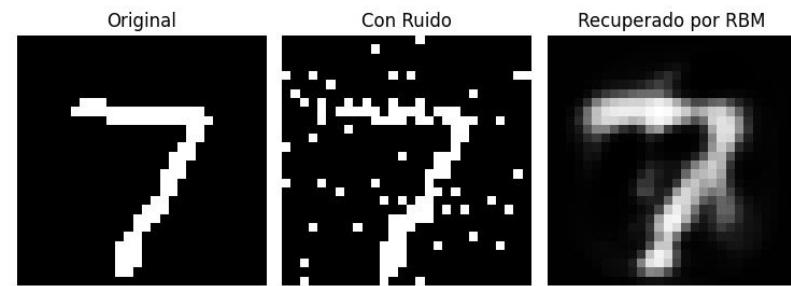
SSIM: 0.5564



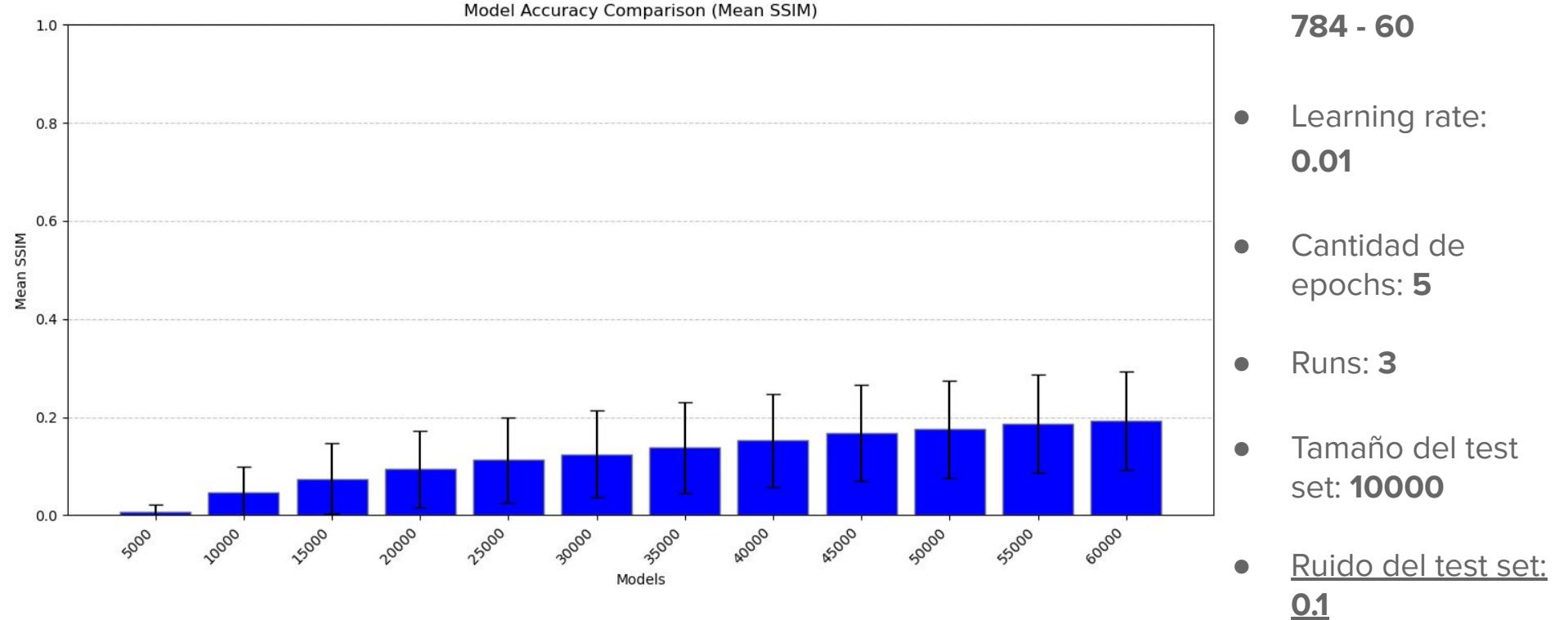
SSIM: 0.0423



SSIM: 0.4320



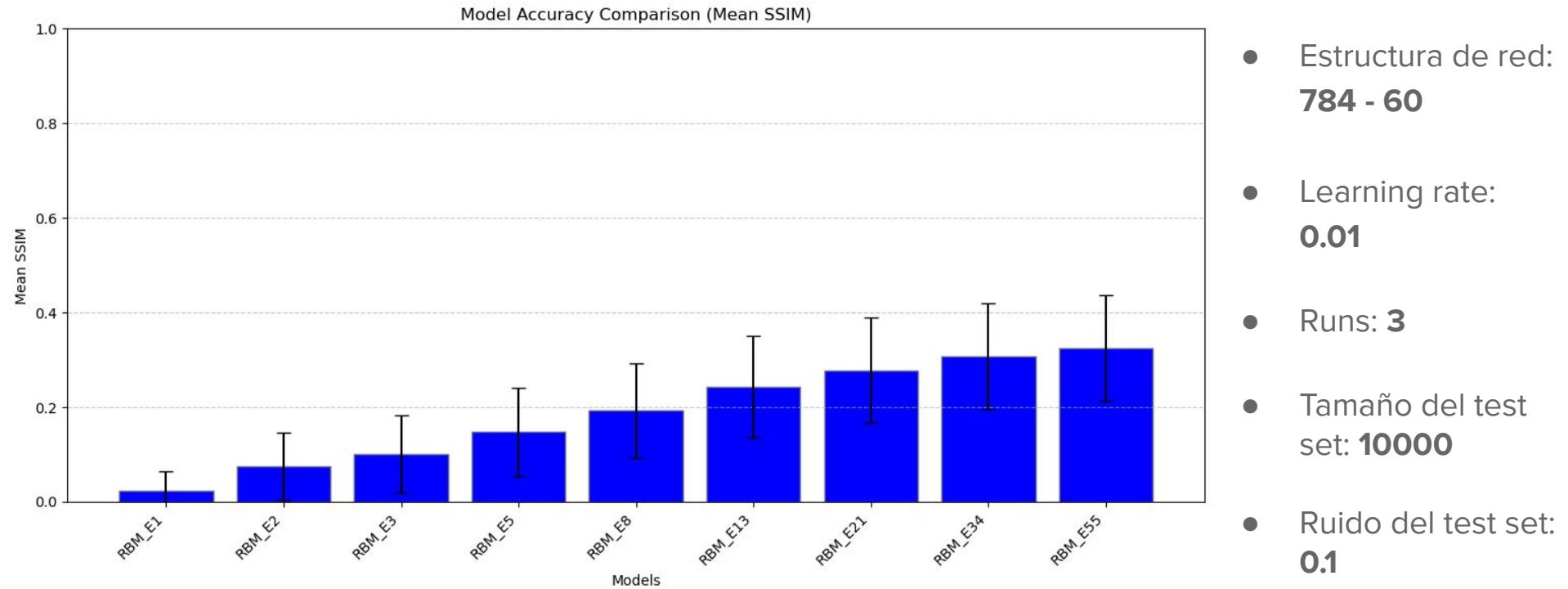
Tamaño del training set



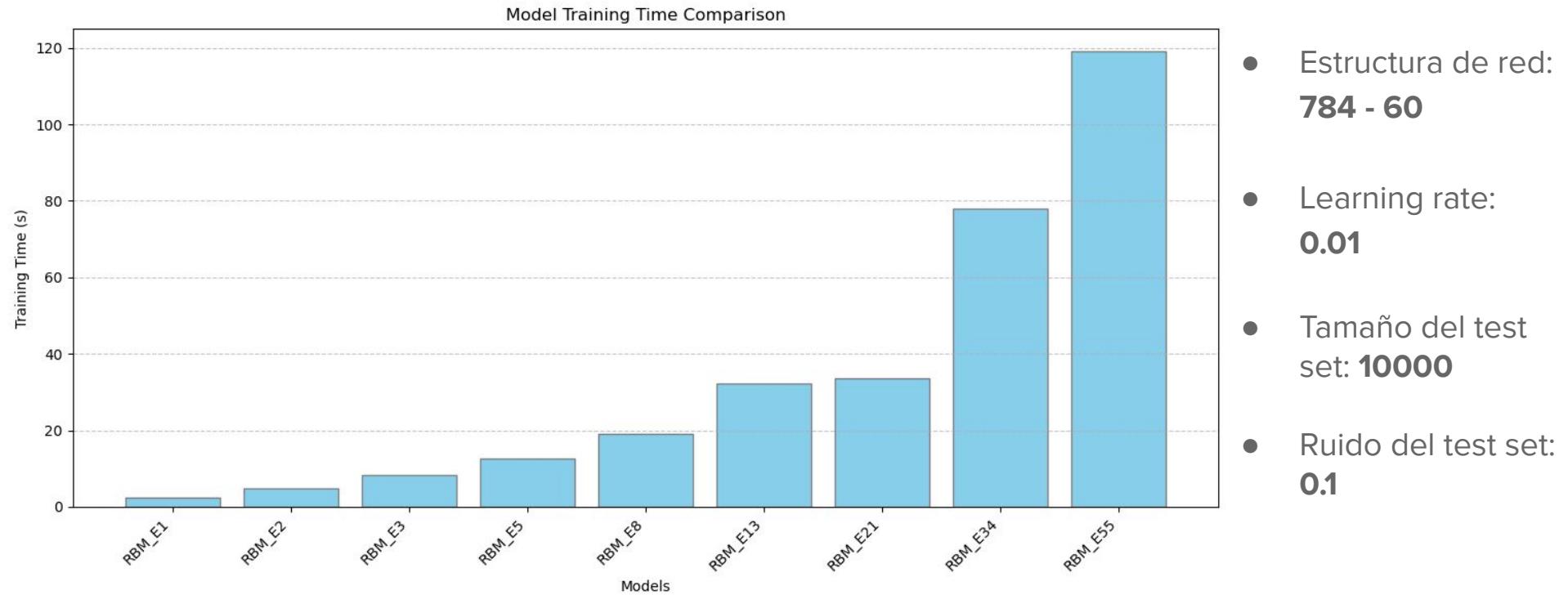
Conclusiones

- Mayor tamaño del training set lleva a mejor accuracy

#Epochs



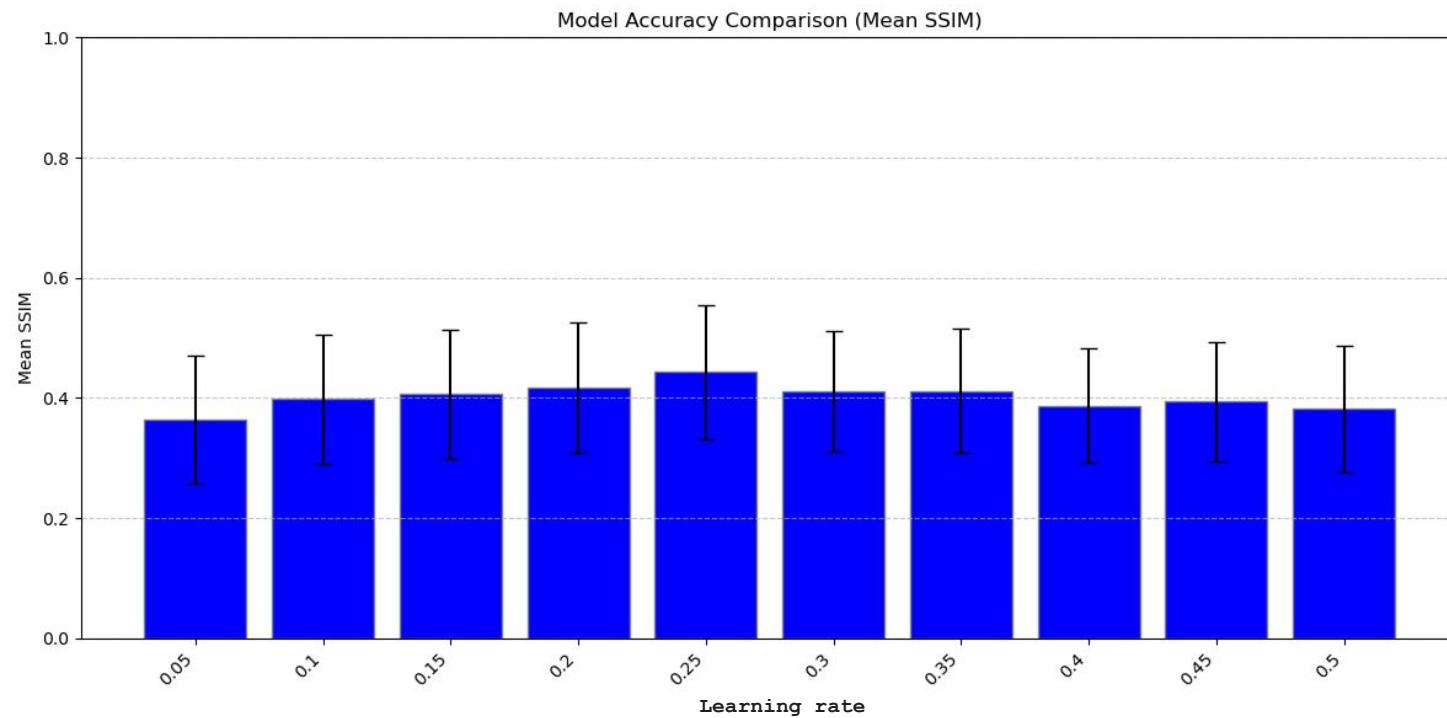
#Epochs



Conclusiones

- Mejor performance a más epochs
- Mayor tiempo de entrenamiento a más epochs

Learning rate



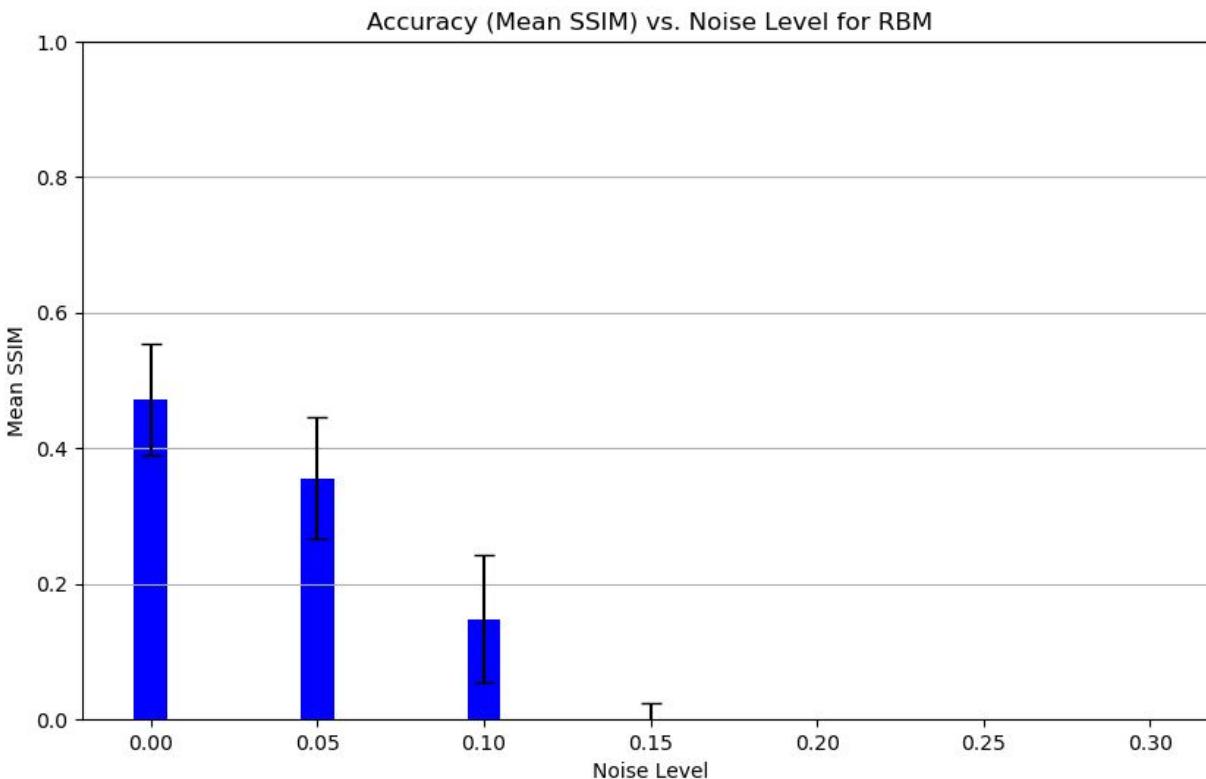
- Estructura de red: **784 - 128**
- Epochs: **5**
- Runs: **3**
- Tamaño del test set: **10000**
- Ruido del test set: **0.1**

Conclusiones

- Mejor performance con learning rate = 0.25 (en este caso, estructura/problema)

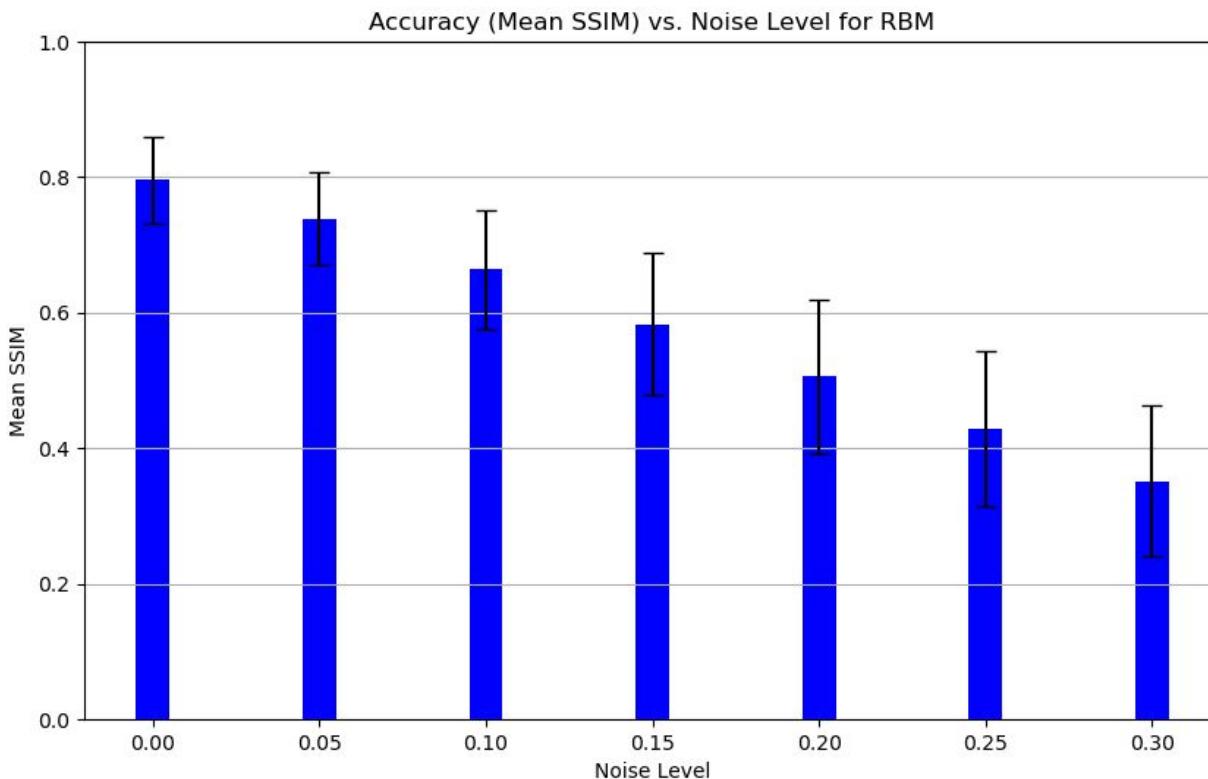
Ruido Binario

Ruido Binario



- Estructura de red:
784 - 60
- Learning rate:
0.01
- Epochs: **5**
- Runs: **3**
- Tamaño del test set: **10000**
- Training time:
12.67s

Ruido Binario



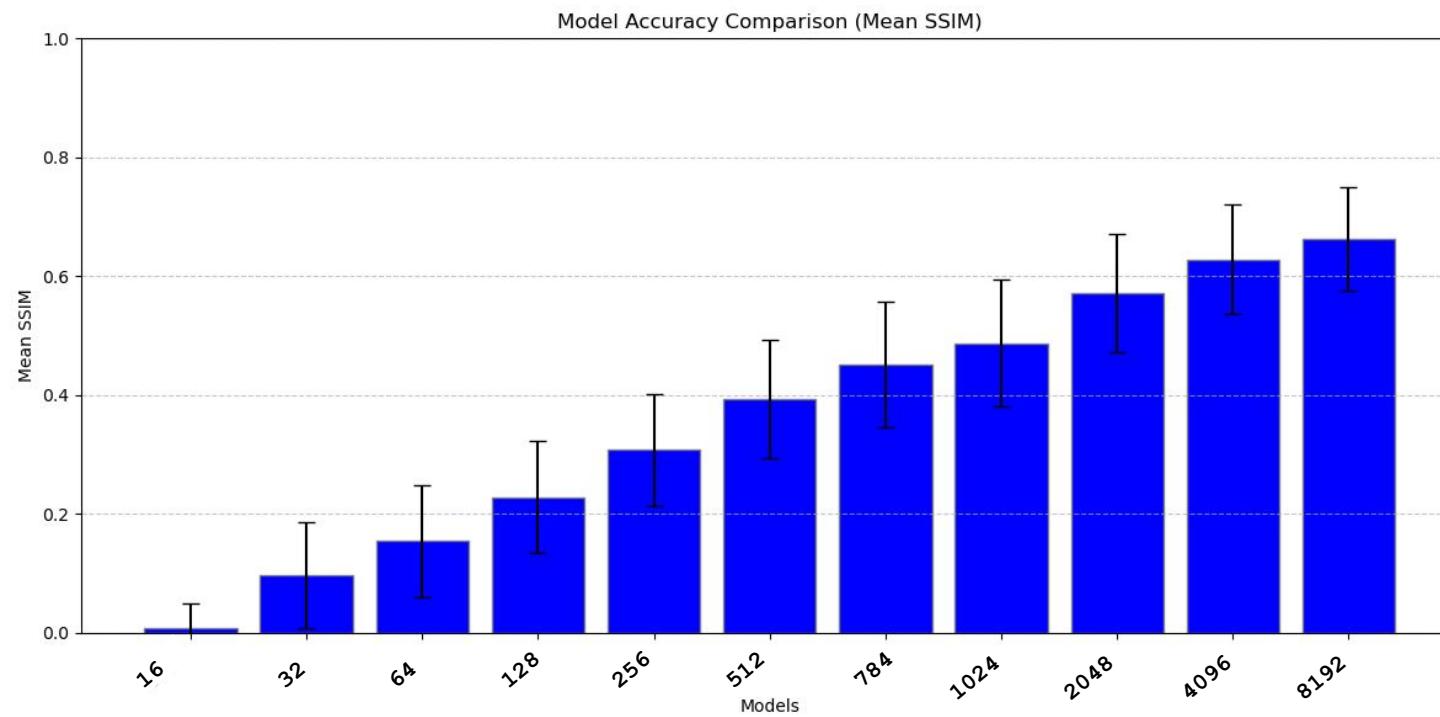
- Estructura de red:
784 - 8192
- Learning rate:
0.01
- Epochs: **5**
- Runs: **3**
- Tamaño del test set: **10000**
- Training time:
947.56s ≈ 16m

Conclusiones

- Peor performance a mayor ruido
- La primera red no es muy resistente al ruido (solamente 10% de ruido reduce la performance casi 40%)
- La segunda red si es más resistente al ruido (un 10% de ruido solamente reduce el accuracy en 10%). Tiene 136 veces más neuronas más que la primera red.

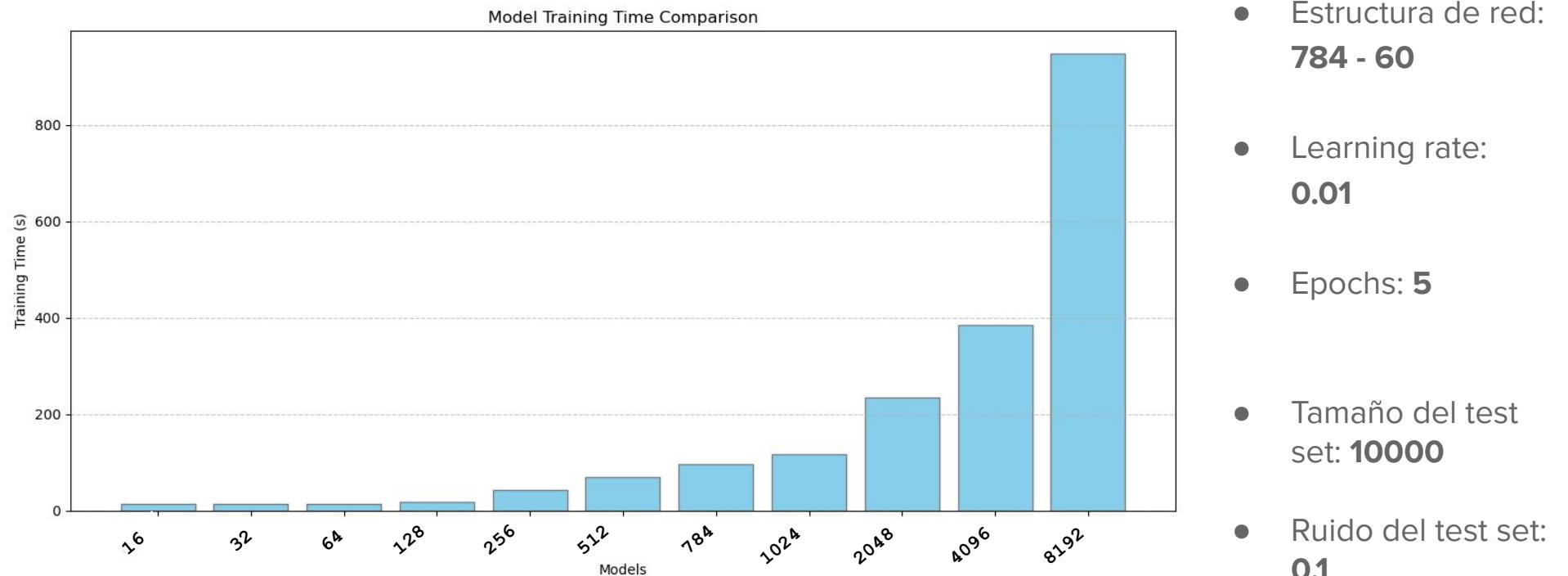
Arquitectura

Arquitecturas



- Estructura de red: **784 - 60**
- Learning rate: **0.01**
- Epochs: **5**
- Runs: **3**
- Tamaño del test set: **10000**
- Ruido del test set: **0.1**

Arquitecturas



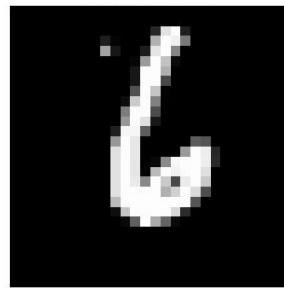
Conclusiones

- Mejor performance con más nodos
- Mayor tiempo de entrenamiento a más nodos
- Notar la diferencia de pendiente entre tiempo y accuracy

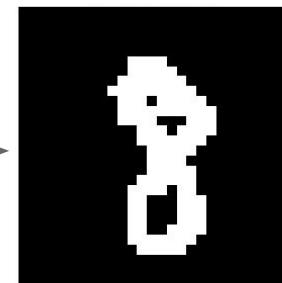
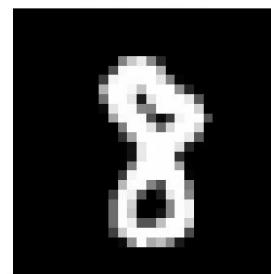
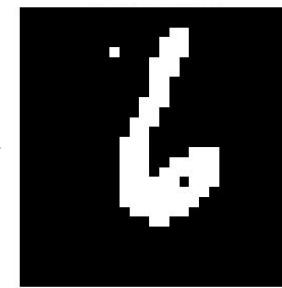
Input binarizado vs. No binarizado

Dato Binarizado vs No Binarizado

No Binarizado



Binarizado



Dato Binarizado vs No Binarizado

No Binarizado

Épocas: 20

Ruido: 7%

Ratio de Aprendizaje: 1%

Binarizado

Imagen Original

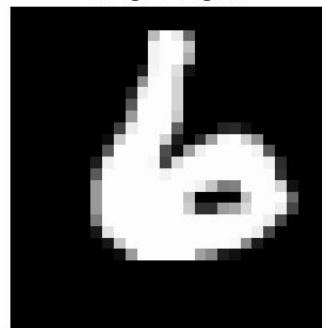


Imagen con Ruido (Nivel 7%)

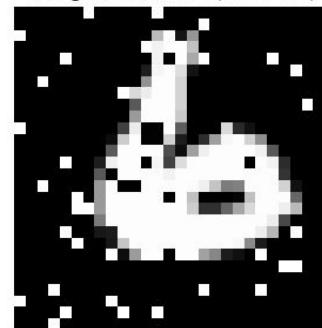


Imagen Reconstruida

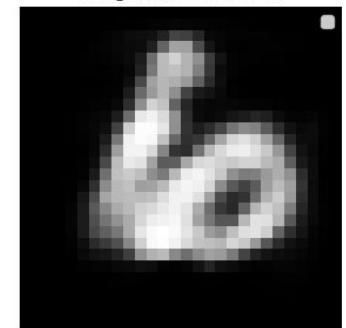


Imagen Original

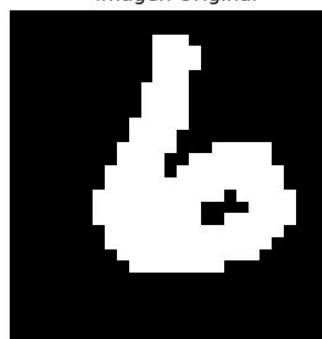
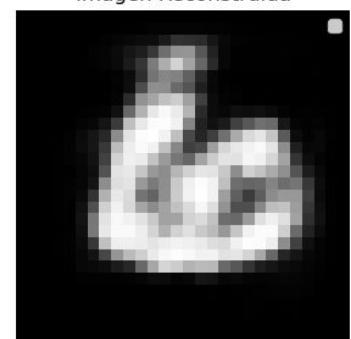


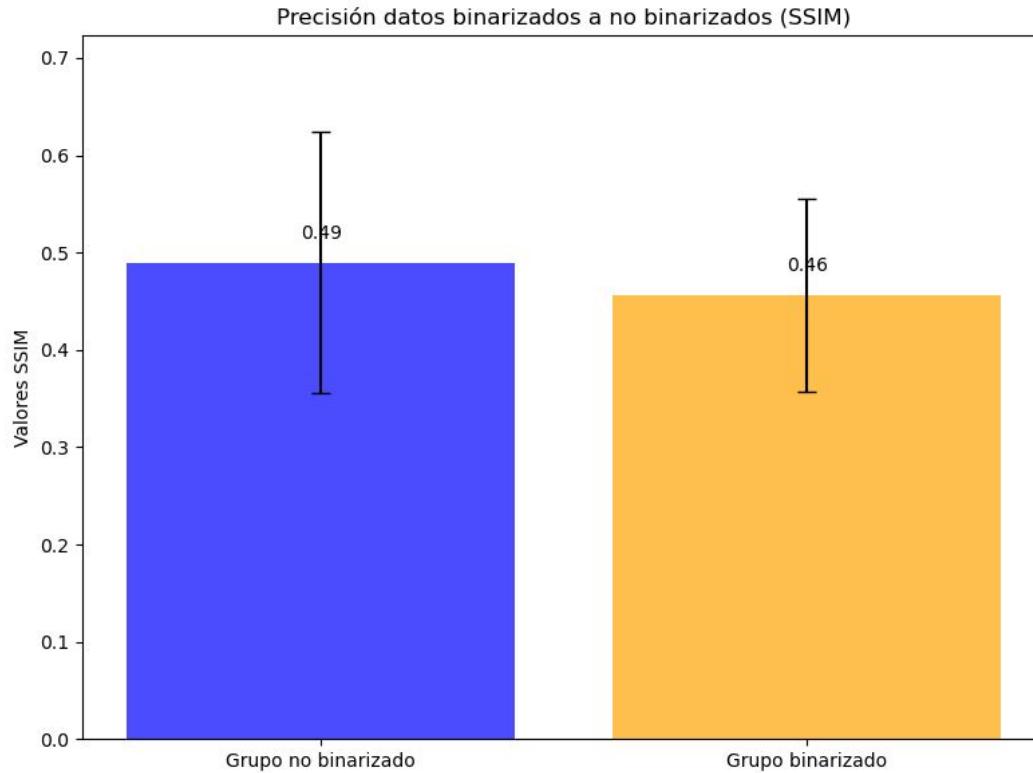
Imagen con Ruido (Nivel 7%)



Imagen Reconstruida



Precisión



Épocas: 20
Ruido: 7%
Ratio de Aprendizaje: 1%

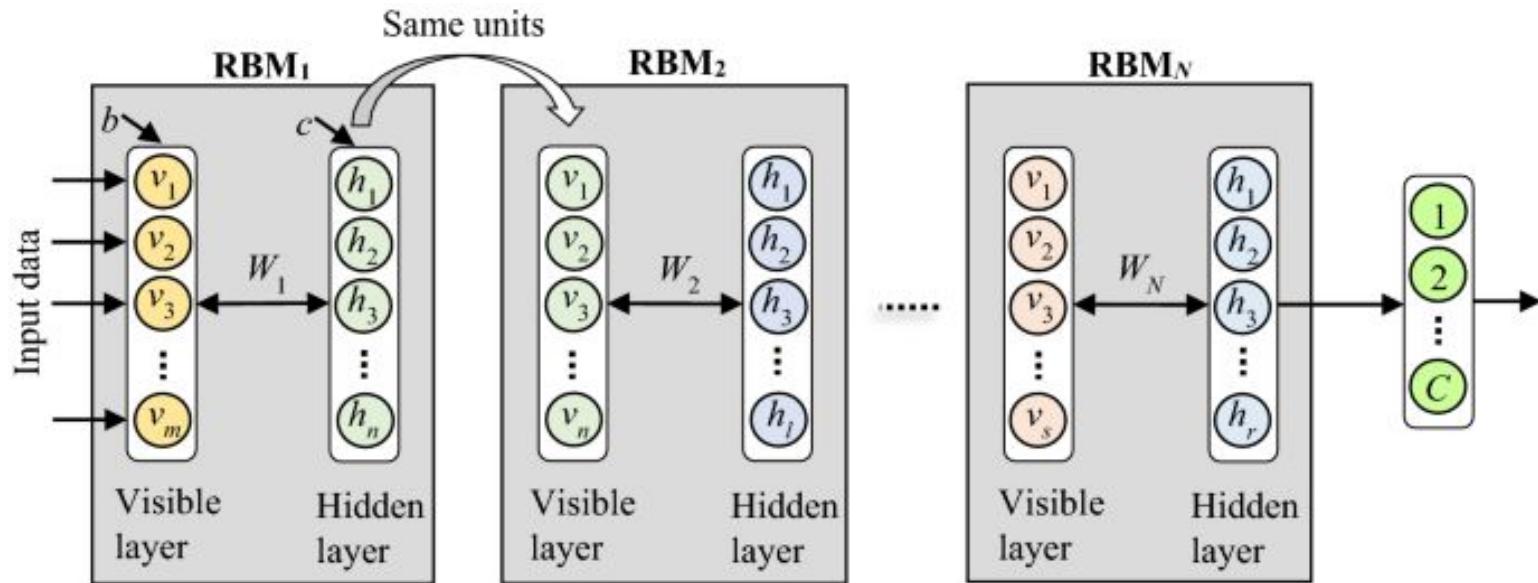
Conclusión

- Pareciera que la binarización de datos no aporta una mejoría sobre datos no binarios para el entrenamiento de la red.

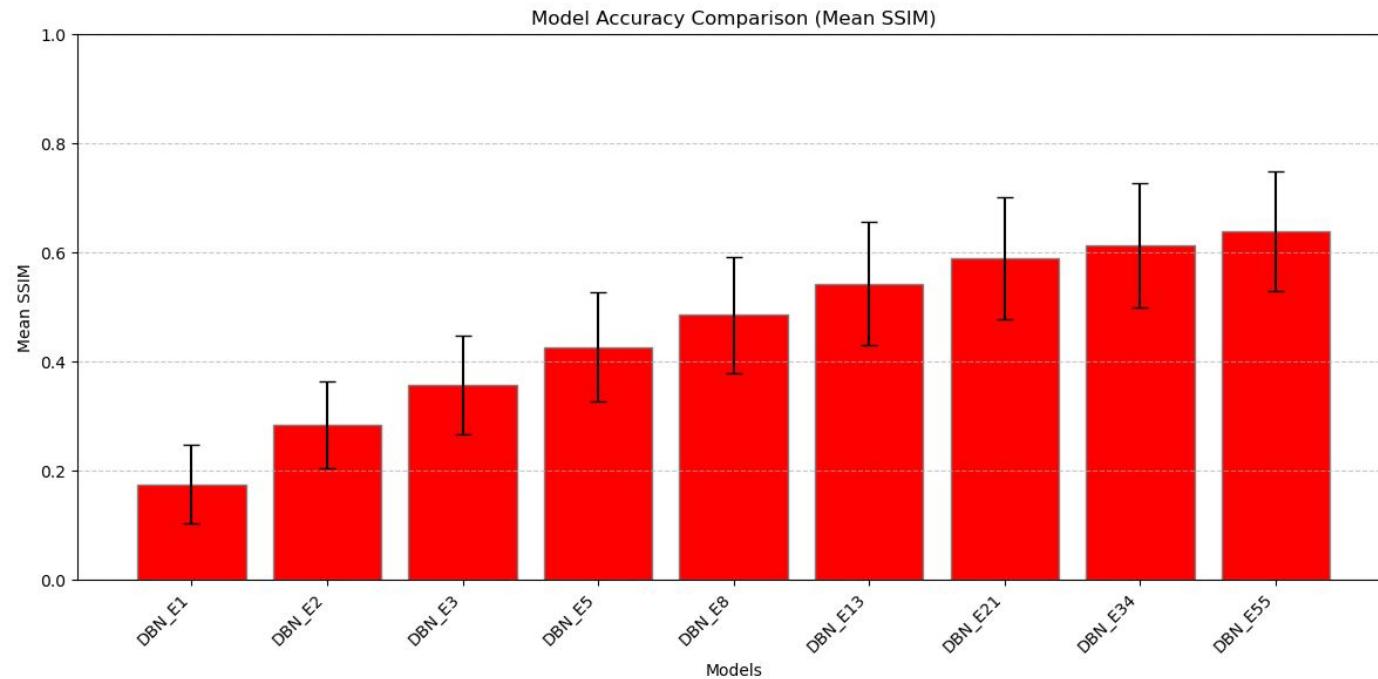
Deep Belief Network (DBN)

Arquitectura

Se entrelazan entre sí muchas RBM donde la capa oculta de una es la capa visible de la otra

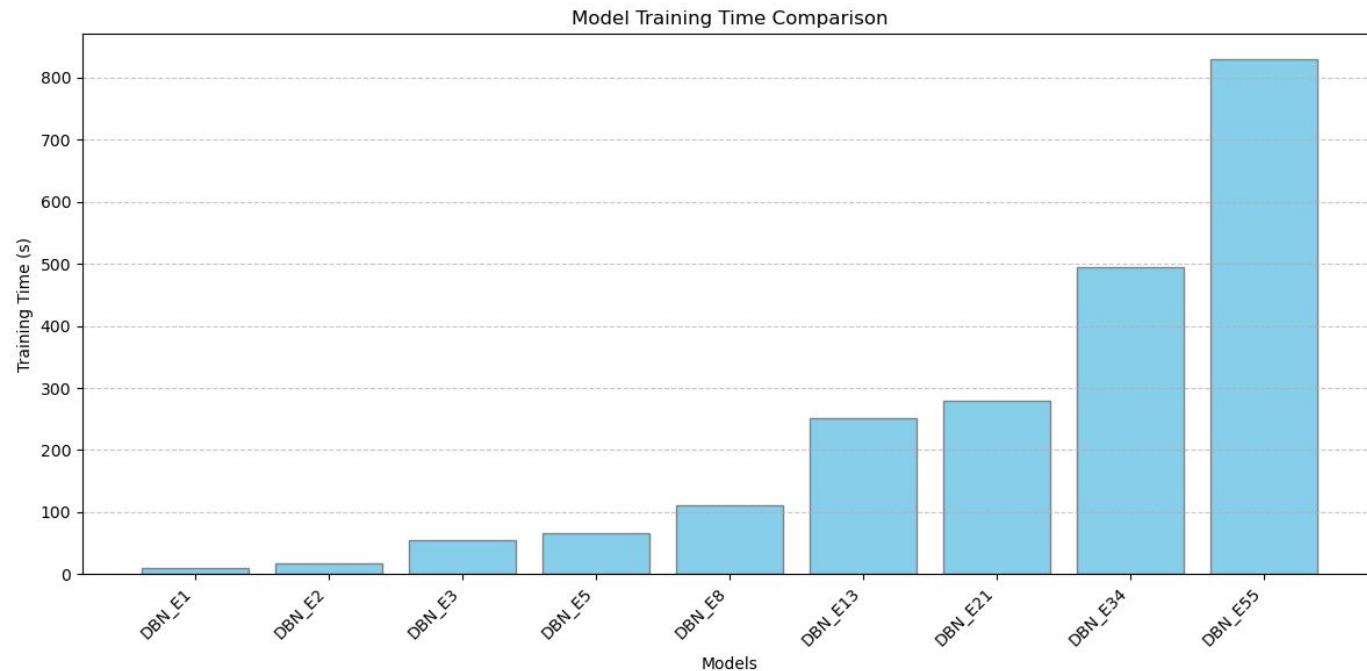


Accuracy vs. Epochs



- Estructura de red: **784 - 500 - 200 - 60**
- Learning rate: **0.01**
- Epochs: **5**
- Runs: **3**
- Tamaño del test set: **10000**
- Ruido del test set: **0.1**

Accuracy vs. Epochs

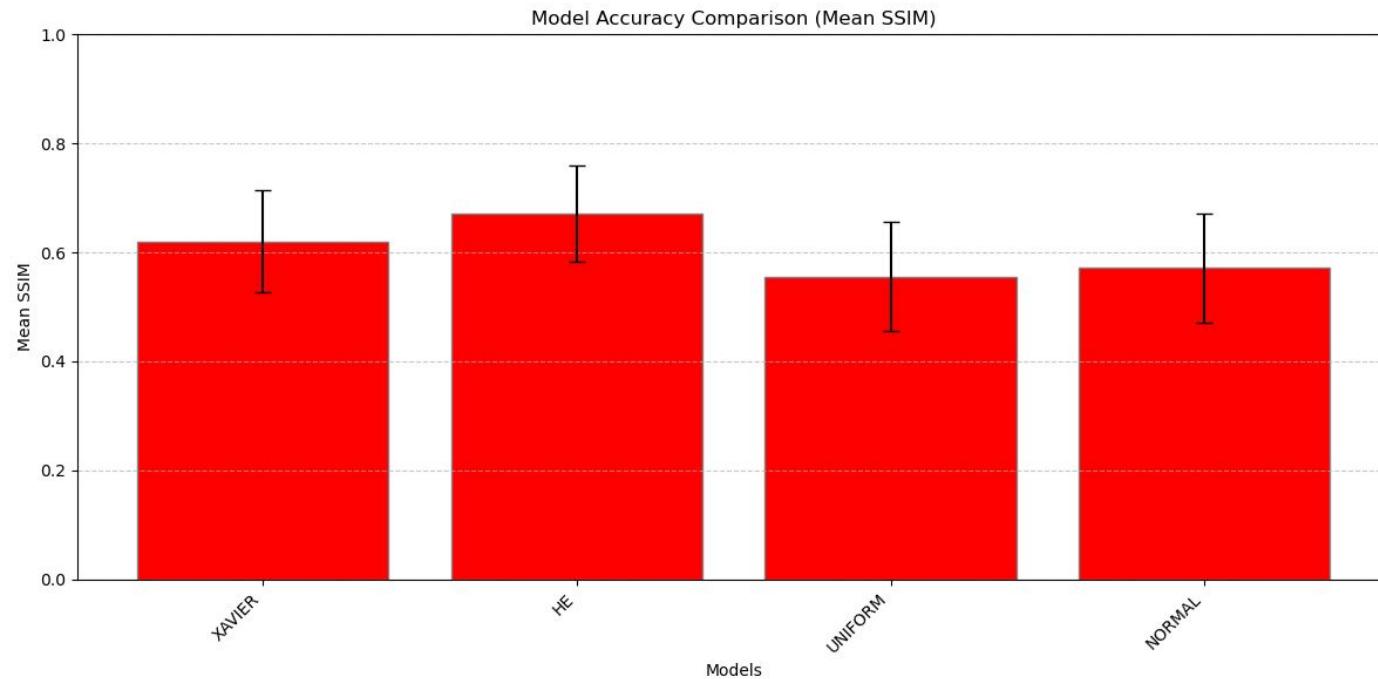


- Estructura de red:
784 - 500 - 200 - 60
- Learning rate: **0.01**
- Epochs: **5**
- Tamaño del test set:
10000
- Ruido del test set: **0.1**

Conclusiones

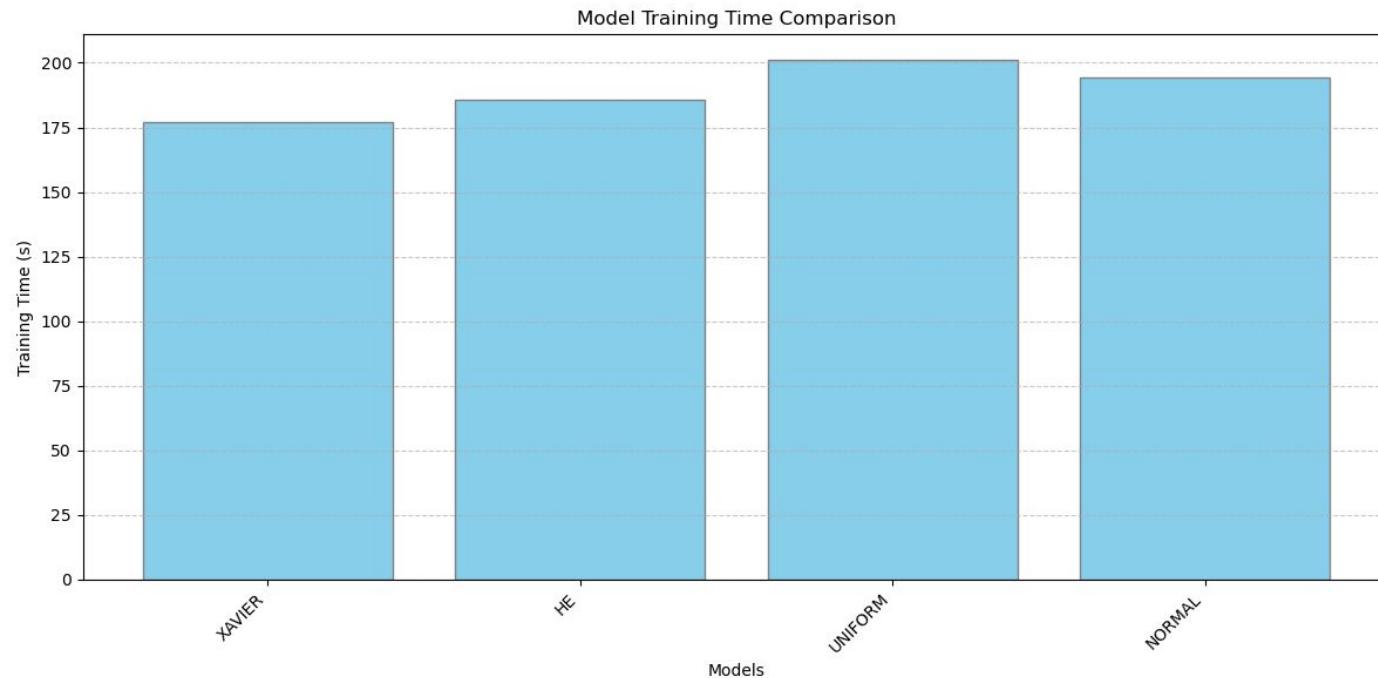
- Mejor performance a más epochas
- Mayor tiempo de entrenamiento a más epochas

Weight Initialization



- Estructura de red: **784 - 2048**
- Learning rate: **0.01**
- Epochs: **5**
- Runs: **3**
- Tamaño del test set: **10000**
- Ruido del test set: **0.1**

Weight Initialization



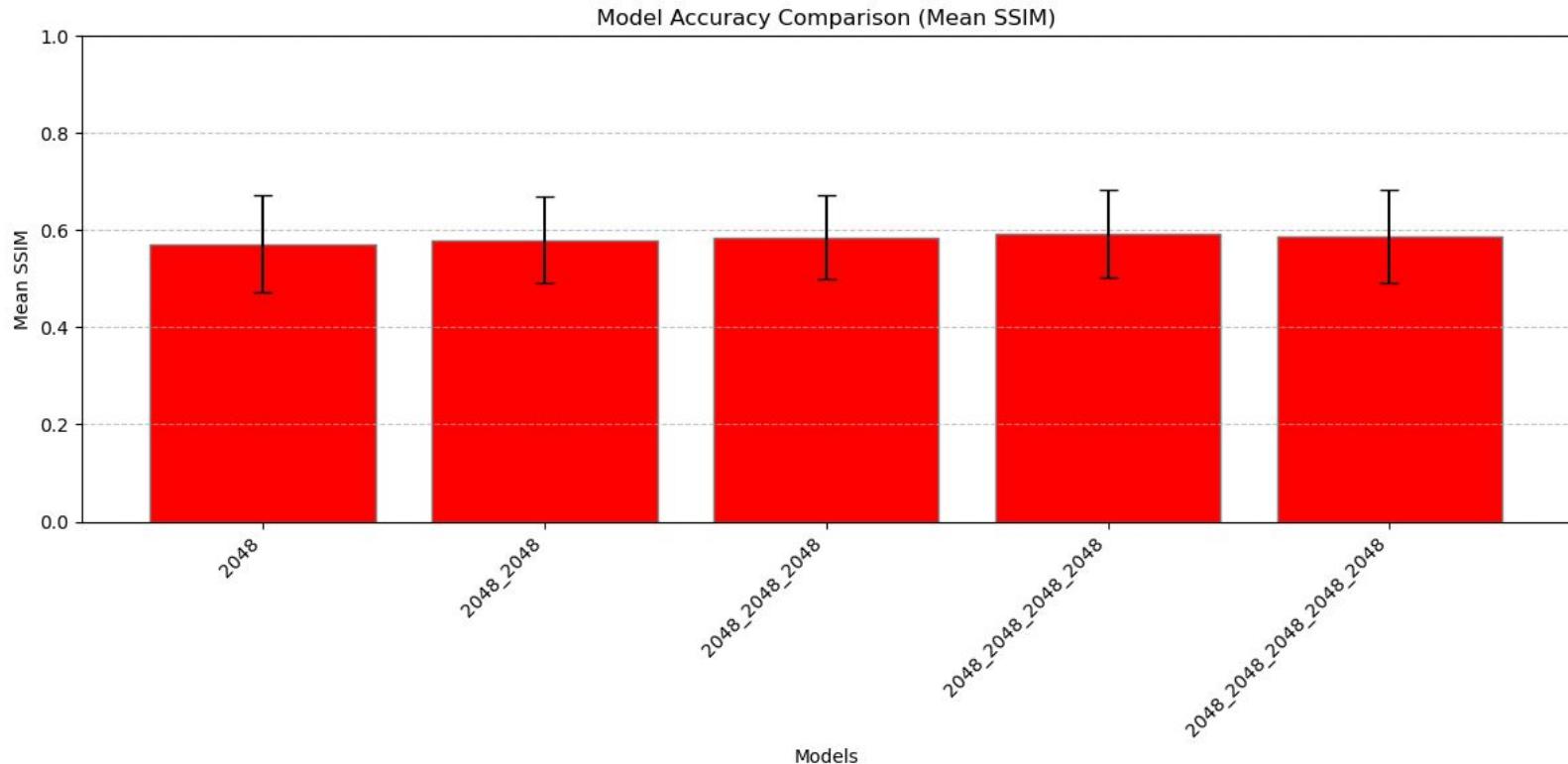
- Estructura de red: **784 - 2048**
- Learning rate: **0.01**
- Epochs: **5**
- Tamaño del test set: **10000**
- Ruido del test set: **0.1**

Conclusiones

- He parece dar los mejores resultados de accuracy en este caso
- Xavier lleva el menor tiempo de entrenamiento en esta run
- En esta red se esperaba que Xavier tenga mejor performance, en este caso no fue así

Arquitectura

Accuracy vs. #RBMs

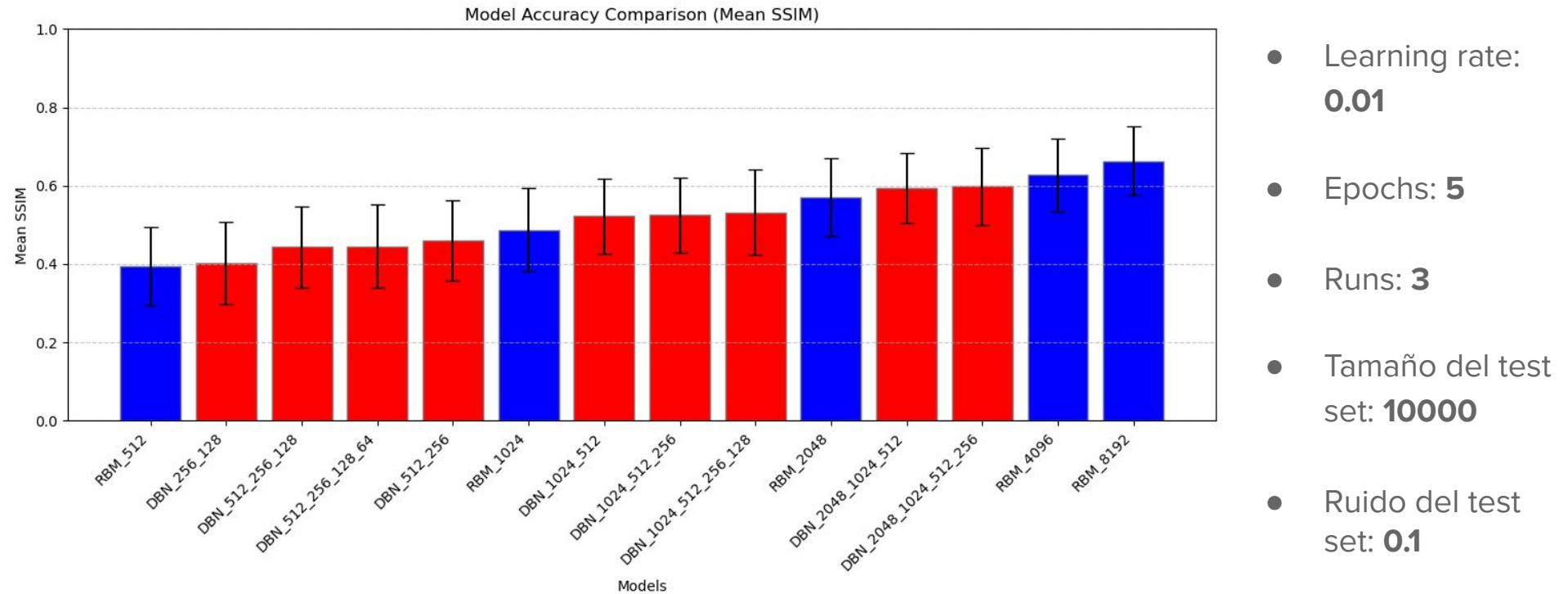


Conclusiones

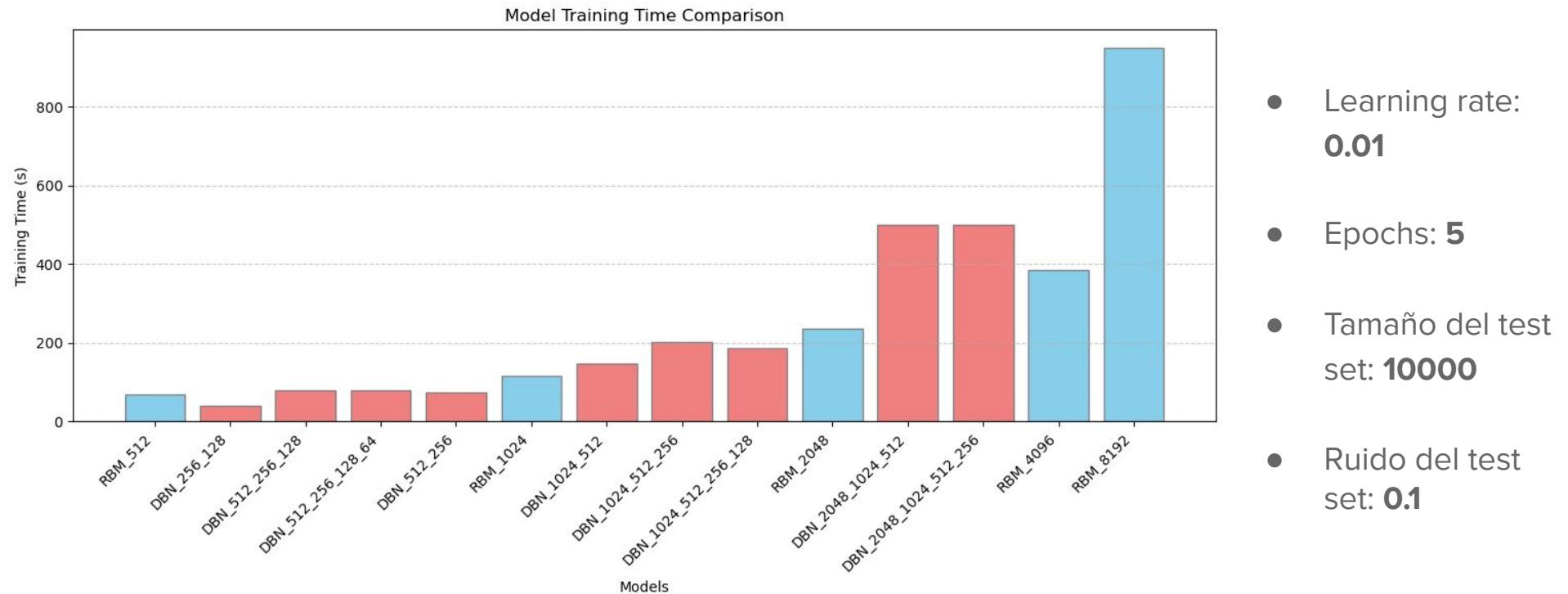
- Más RBMs aumentan muy marginalmente el accuracy en este caso
- Parece que el problema no es tan complejo como para necesitar varias capas (con esta cantidad de nodos), ya reconoce las estructuras con una sola RBM de ese tamaño
- DBN funciona casi igual a RBM cuando la primer RBM es suficientemente grande (en este caso)

DBN vs. RBM

Accuracy vs. Arquitecturas



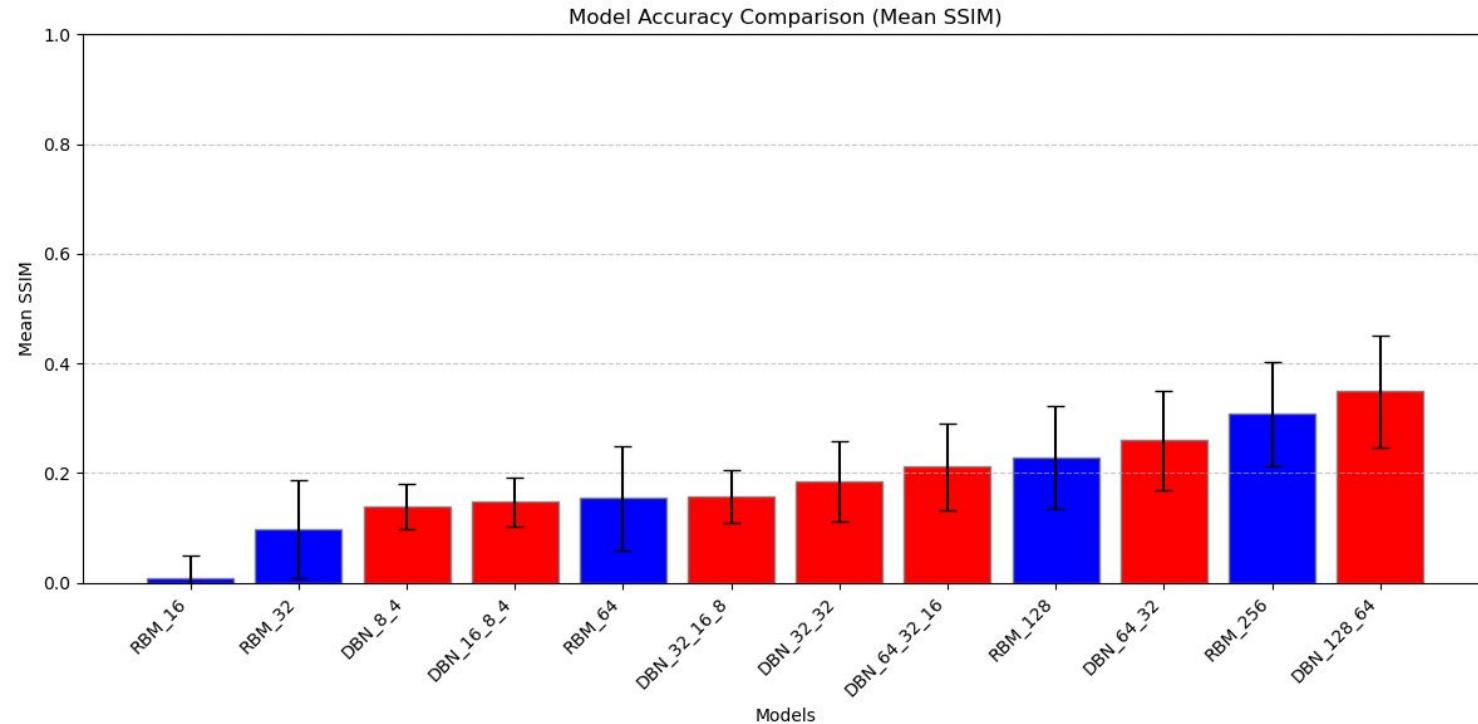
Accuracy vs. Arquitecturas



Conclusiones

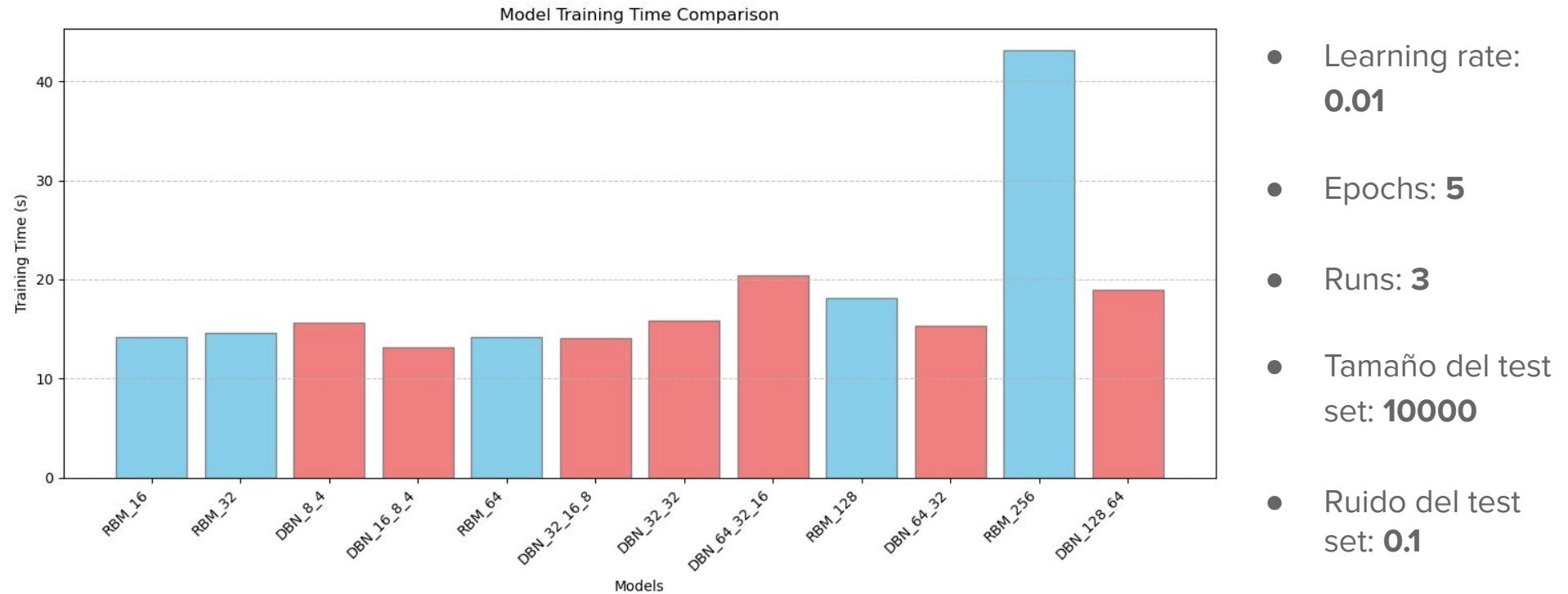
- A más cantidad de nodos en la primera capa mayor accuracy (cuando es suficientemente grande)
- Hay casos en los cuales una DBN de tamaño menor (menos nodos) tarda más tiempo que una RBM y tiene peor accuracy (ej. RBM_4096 vs. DBN_2048_1024_512_256)
- Pero también hay casos donde lo opuesto ocurre... (ej. RBM_512 vs. DBN_256_128)

Accuracy vs. Arquitecturas



- Learning rate: **0.01**
- Epochs: **5**
- Runs: **3**
- Tamaño del test set: **10000**
- Ruido del test set: **0.1**

Accuracy vs. Arquitecturas



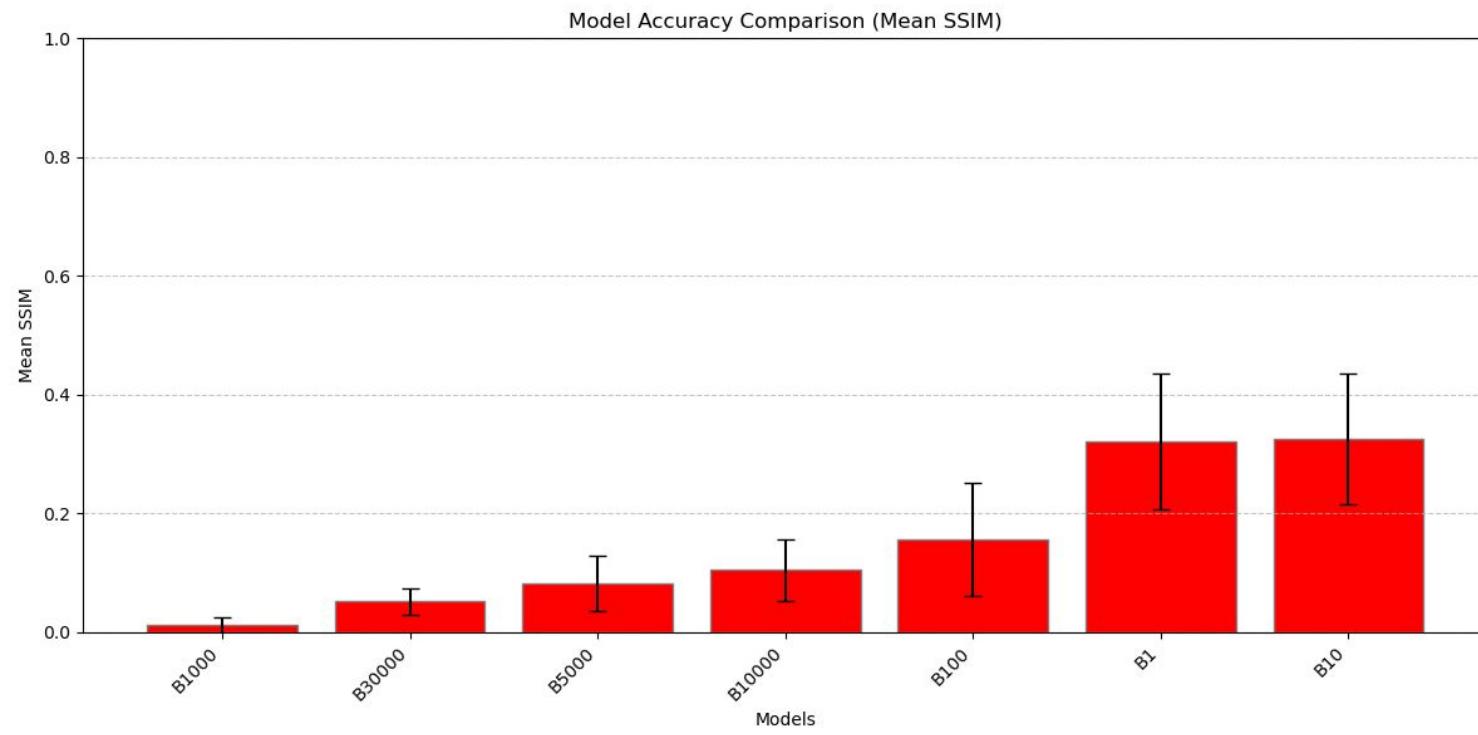
Conclusiones

- DBN funciona mejor que RBM en los casos donde la red no termina de “entender” las estructuras (redes chicas para este caso)
- DBN puede entrenarse más rápido y tener mejor accuracy (ej. RMB_256 vs. DBN_128_64)
- DBNs con estructuras que tienen menos neuronas pueden tener mejor accuracy que RBMs con más neuronas (ej. RMB_256 vs. DBN_128_64)

**Gracias por su
atención :)**

Anexo

Batching



- Estructura de red: **784 - 64**
- Learning rate: **0.01**
- Epochs: **5**
- Runs: **3**
- Tamaño del test set: **10000**
- Ruido del test set: **0.1**