Autoencoders

- Para reemplazar PCA en un contexto en el cual necesitamos un modelo end-to-end totalmente diferenciable podemos usar autoencoders para reducir la dimensionalidad.
- Los autoencoders son redes neuronales con una estructura particular que nos permiten aprender una representación de menor dimensionalidad de los datos de entrada. Este espacio latente es una representación densa de los datos de entrada que puede ser usada para reconstruir los datos originales.
- En este caso sabemos que PCA da buenos resultados para reducir la dimensionalidad de los datos, por lo que vamos a comparar primero PCA con autoencoders con activaciones no lineales y luego con autoencoders con activaciones lineales. La métrica que vamos a utilizar es comparar el error de reconstrucción entre las señales originales y las reconstruidas.

PCA

- Utilizamos la implementación de PCA que habiamos hecho anteriormente.

```
1 using PlutoUI
```

Multiple expressions in one cell.

How would you like to fix it?

- Split this cell into 12 cells, or
- Wrap all code in a *begin ... end* block.

```
1  using MultivariateStats
2  using DataFrames
3  using CSV
4  using Statistics
5  using Flux
6  using Plots
7  using Random
8  using IterTools: ncycle
9  using ProgressMetera
10 using LinearAlgebra
11 using Measures
12 include("../1-GeneracionDatos/Parameters.jl");
```

UndefVarError: `CSV` not defined

Stack trace

Here is what happened, the most recent locations are first:

1. *This cell: line 1*

```
1 signalsDF = transpose(Matrix(CSV.read("C:/Users/Propietario/OneDrive/Escritorio/ib/Tesis_V1/MLonNMR/1-GeneracionDatos/Data/SimpleSignalHahn_TE_1_G_8.73e-7.csv", DataFrame)))
```

```
1 signalsDF = transpose(Matrix(CSV.read("C:/Users/Propietario/OneDrive/Escritorio/ib/Tesis_V1/MLonNMR/1-GeneracionDatos/Data/SimpleSignalHahn_TE_1_G_8.73e-7.csv", DataFrame)))
```

Multiple expressions in one cell.

How would you like to fix it?

- Split this cell into 5 cells, or
- Wrap all code in a *begin ... end* block.

```
1  column_lcm = collect(lcms)
2  column_sigma = collect(σs)
3
4  pdistparamsDF = zeros(size(signalsDF)[2], 2)
5
6  for (i, lcm) in enumerate(column_lcm)
7      for (j, sigma) in enumerate(column_sigma)
8          pdistparamsDF[(i - 1) * length(σs) + j, 1] = sigma
9          pdistparamsDF[(i - 1) * length(σs) + j, 2] = lcm
10     end
11 end
12
13 pdistparamsDF = DataFrame(pdistparamsDF, [:sigma, :lcm]);
```

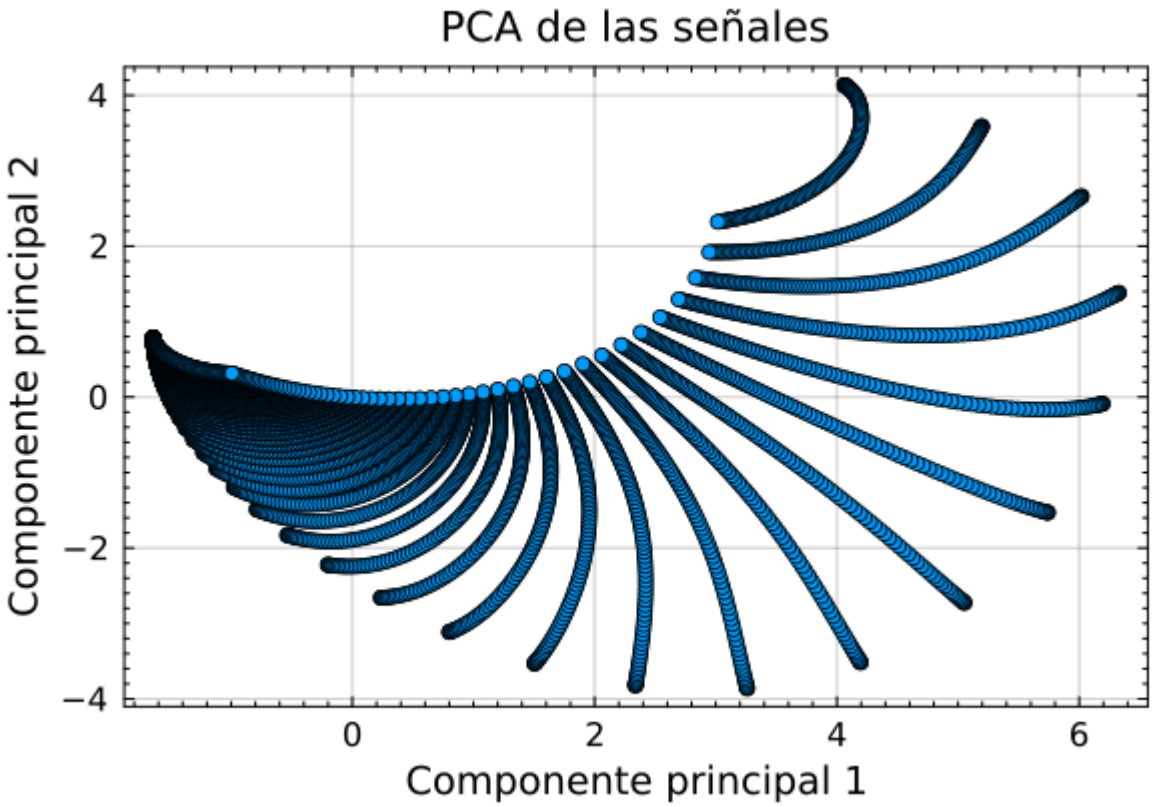
centerData (generic function with 1 method)

```julia
# En cada columna tenmos los datos de las señales, centramos estas columnas para que tengan media 0
function centerData(matrix)
    """Función que centra los datos de las columnas de una matriz para que tengan media 0
    Parametros
        matrix: matriz con los datos a centrar
    Retorna
        centered_data: matriz con los datos centrados
    """
    col_means = mean(matrix, dims=1)
    centered_data = matrix .- col_means
    return centered_data, col_means
end
```

dataPCA (generic function with 1 method)

```julia
# Función que realiza PCA sobre los datos de entrada y grafica la varianza explicada por cada componente principal
function dataPCA(dataIN)
    """Función que realiza PCA sobre los datos de entrada y grafica la varianza explicada por cada componente principal

    Parametros
        dataIN: matriz con los datos a los que se les va a realizar PCA
    Retorna
        reduced_dataIN: datos reducidos por PCA
        pca_model: modelo de PCA que se puede usar para reconstruir los datos originales, además contiene información sobre los componentes principales
    """

    # Primero centramos los datos
    dataIN_C, _ = centerData(dataIN)

    # Esto ya hace PCA sobre la matriz dada donde cada observación es una columna de la matriz
    pca_model = fit(PCA, dataIN_C, maxoutdim=3)

    # Esta instancia de PCA tiene distintas funciones como las siguientes

    #projIN = projection(pca_model) # Proyección de los datos sobre los componentes principales

    # Vector con las contribuciones de cada componente (es decir los autovalores)
    pcsIN = principalvars(pca_model)

    # Obtenemos la variaza en porcentaje para cada componente principal
    explained_varianceIN = pcsIN / sum(pcsIN) * 100

    reducedIN = MultivariateStats.transform(pca_model, dataIN_C)

    return reducedIN, pca_model
end
```

- En PCA es mejor centrar los datos para que tengan media 0 antes de aplicarlo, esto es algo que también vamos a repetir en los autoencoders.



- Obtenemos el mismo resultado de siempre.

Error de Reconstrucción

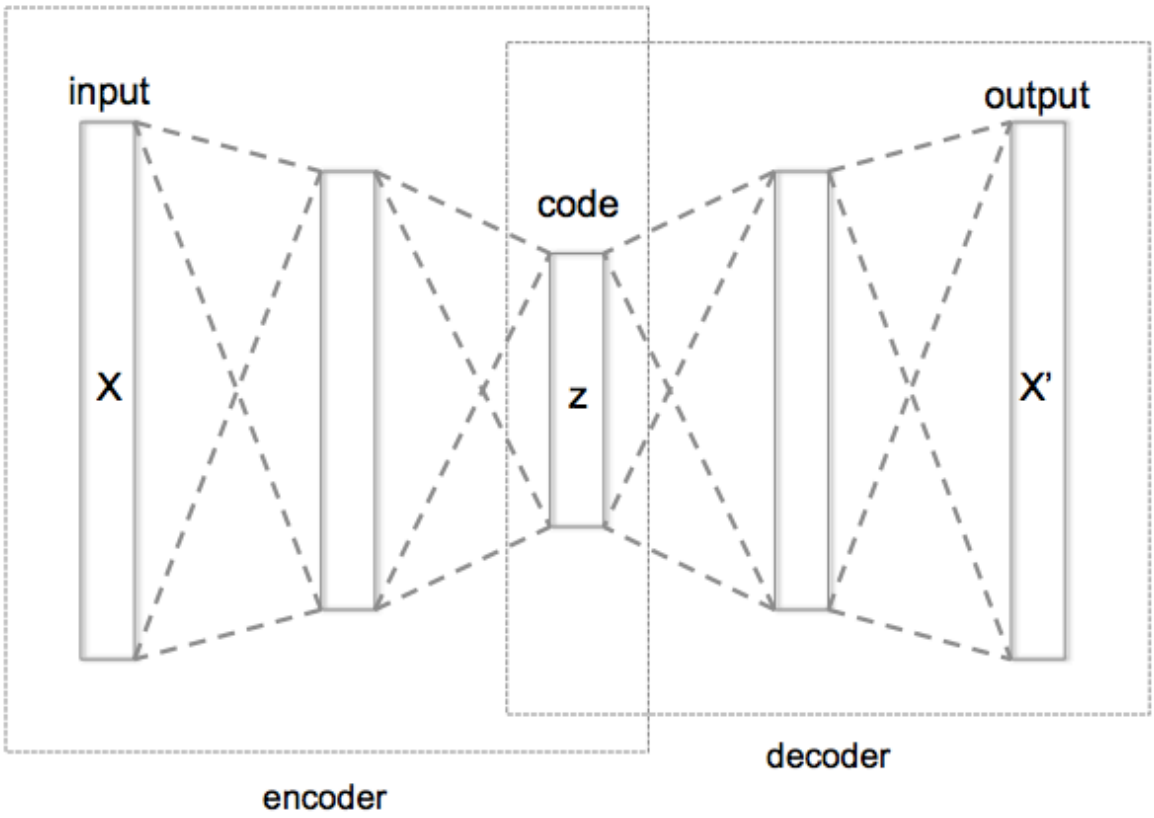
- Usamos la inversa de la matriz que usamos para reducir la dimensionalidad con PCA y así reconstruir los datos originales y ver si se pierde información en este proceso de reconstrucción.
- La métrica que vamos a usar de error de reconstrucción es la raíz del error cuadrático medio (RMSE) calculada como $\sqrt{\frac{1}{N}\sum_{i=1}^{N}\frac{1}{M}\sum_{j=1}^{M}(S_j-\hat{S}_j)_i^2}$ donde $(S_j-\hat{S}_j)_i^2$ es la resta cuadrática entre de los puntos j de la señal i verdaderos contra la reconstrucción. N es la cantidad de señales y M el número de puntos en cada señal

- Error MSE de reconstrucción de PCA: 4.7125315997926665e-5
- Error RMSE de reconstrucción de PCA: 0.006864788124765882

Autoencoders

- A diferencia de PCA, los autoencoders pueden aprender representaciones no lineales de los datos, sin embargo hay un gran paralelismo entre PCA y autoencoders cuando estos tienen una sola capa oculta con una capa de activación lineal. Como dijimos antes los autoencoders tienen una estructura particular que consiste en dos redes neuronales un encoder y un decoder. El encoder toma los datos de entrada como M dimensiones y los reduce a un espacio

latente de menor dimensionalidad. El decoder toma el espacio latente y lo reconstruye a las M dimensiones originales. En este caso el espacio latente que vamos a querer aprender es de 3 dimensiones que es lo que nos da PCA por defecto sin perder casi nada de información.



- Vamos a explorar tres autoencoders distintos uno simple con una única capa oculta, otro con 4 capas ocultas y otro con 6 capas ocultas. En todos los casos vamos a comparar autoencoders con activaciones lineales y no lineales.
- Como vimos que PCA funciona bien con 3 dimensiones la dimension del espacio latente en los autoencoders va a ser de 3.
- Para evitar el overfitting vamos a utilizar regularización L2 en los pesos de las capas ocultas, además usamos early stopping que consiste en detener el entrenamiento cuando el error de validación deja de disminuir durante 100 épcas.
- El error de reconstrucción lo vamos a calcular de la misma forma que en PCA para poder comparar los resultados.
- También los datos van a ser comparados con la señal centrada

Autoencoder Simple

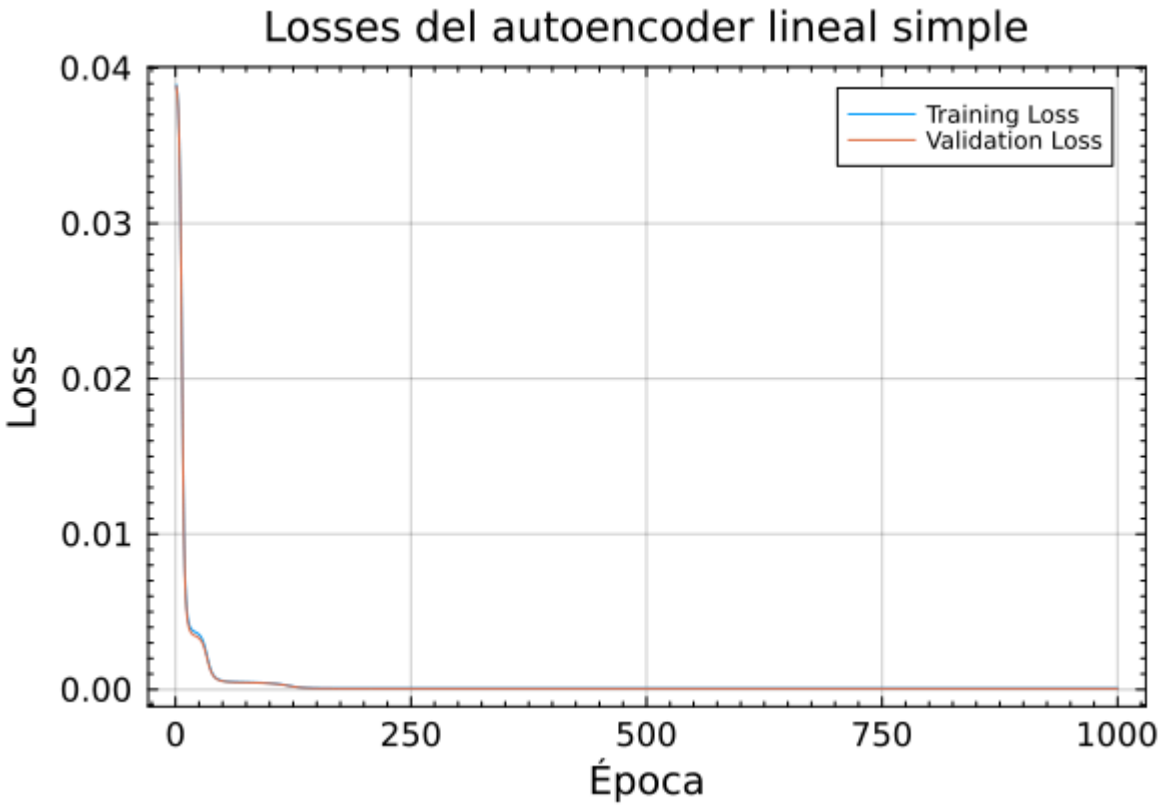
- Este autoencoder tiene una sola capa oculta con 3 neuronas para aprender la representación de menor dimensionalidad de las señales de entrada.

Con activaciones lineales

> **Multiple expressions in one cell.**
>
> How would you like to fix it?
>
> - Split this cell into 24 cells, or
> - Wrap all code in a *begin ... end* block.

```
# Split en training, validation y test
n_signals = size(signalsDF_C, 2)
n_train = Int(floor(n_signals*0.7))
n_val = Int(floor(n_signals*0.15))
n_test = n_signals - n_train - n_val

train_signals = Float32.(Matrix(signalsDF_C[:, 1:n_train]))
val_signals = Float32.(Matrix(signalsDF_C[:, n_train+1:n_train+n_val]))
test_signals = Float32.(Matrix(signalsDF_C[:, n_train+n_val+1:end]))

train_params = pdistparamsDF[1:n_train, :]
val_params = pdistparamsDF[n_train+1:n_train+n_val, :]
test_params = pdistparamsDF[n_train+n_val+1:end, :];

# Number of dimension of the unreduced data
n_times = size(train_signals, 1)

# Autoencoder simple
encoderSimpleLineal = Chain(Dense(n_times, 3, identity, bias = false))
decoderSimpleLineal = Chain(Dense(3, n_times, identity, bias = false))
autoencoderSimpleLineal = Chain(encoderSimpleLineal, decoderSimpleLineal)

# Destructure in parameters and NN structure
s_params, s_re = Flux.destructure(autoencoderSimple)

# Dataloader and mini-batching for traininig
loader = Flux.Data.DataLoader((train_signals, train_signals), batchsize=50,
shuffle=true)

# Initial learning rate
lr = 0.001

# Optimizer used
optim = Flux.setup(Flux.AdamW(lr), autoencoderSimple)

# Number of epochs
num_epochs = 500
# Patience for reducing learning rate and checking if the val loss has decresed
patience_epochs = 100

# Loss function
function loss_re(x, y)
    return Flux.mse(s_re(x), y)
end

```

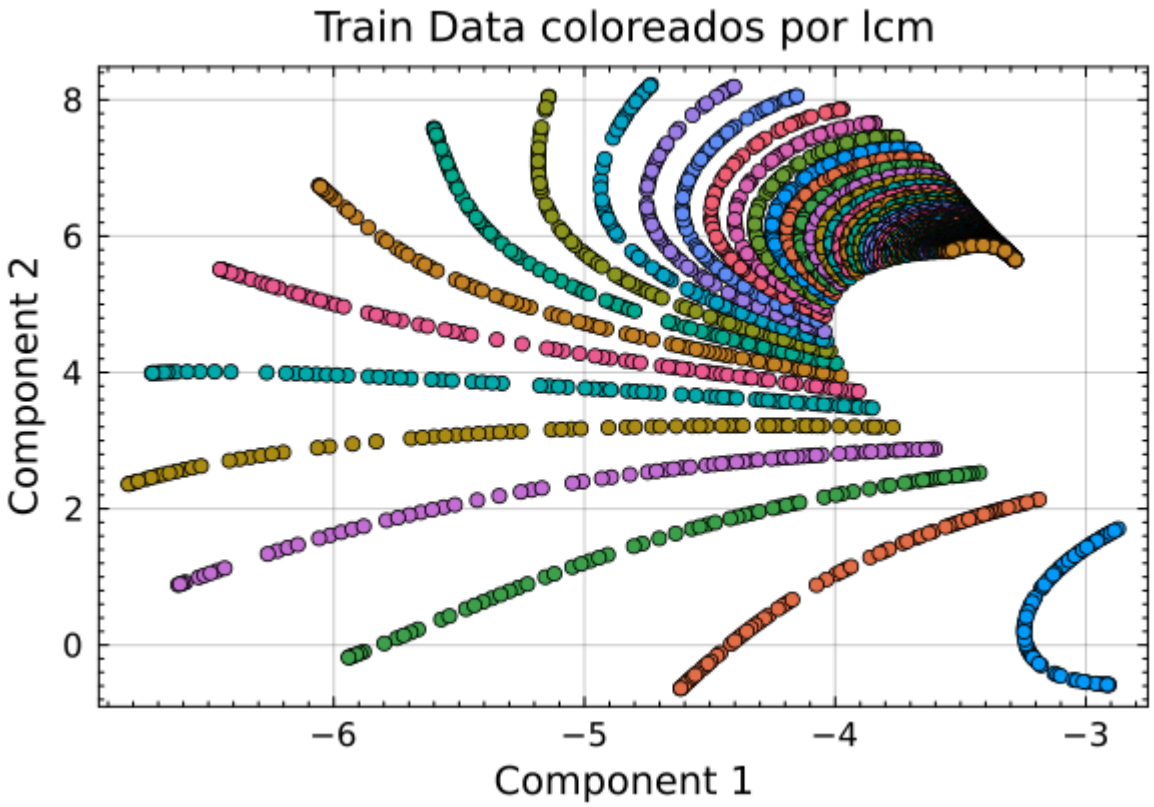
```julia
# Losses history
lossesSimple = []
lossesSimpleVal = []

# Training loop
for epoch in 1:num_epochs
    for (x, y) in loader
        loss, grads = Flux.withgradient(autoencoderSimple) do m
            Flux.mse(m(x), y)
        end
        Flux.update!(optim, autoencoderSimple, grads[1])
    end
    actual_loss = Flux.mse(autoencoderSimple(train_signals), train_signals)
    actual_loss_val = Flux.mse(autoencoderSimple(val_signals), val_signals)

    # Saving best model
    if epoch > 1  && (actual_loss_val < minimum(lossesSimpleVal))

        encoder_params, encoder_re = Flux.destructure(autoencoderSimple[1])
        decoder_params, decoder_re = Flux.destructure(autoencoderSimple[2])

        df_encoder = DataFrame(reshape(encoder_params, length(encoder_params), 1), :auto)
        df_decoder = DataFrame(reshape(decoder_params, length(decoder_params), 1), :auto)

        CSV.write("C:/Users/Propietario/OneDrive/Escritorio/ib/Tesis_V1/MLonNMR/2-2-Autoencoders/Models/minAE_PCAParamsE.csv", df_encoder)
        CSV.write("C:/Users/Propietario/OneDrive/Escritorio/ib/Tesis_V1/MLonNMR/2-2-Autoencoders/Models/minAE_PCAParamsD.csv", df_decoder)
    end

    push!(lossesSimple, actual_loss)
    push!(lossesSimpleVal, actual_loss_val)
    println("Epoch: ", epoch, " Loss: ", actual_loss, " Loss Val: ", actual_loss_val)

    # Early stopping and decreasing learning rate after patience epochs
    if epoch % patience_epochs == 0
        Flux.adjust!(optim, lr * 0.1)
        loss_val_prev = lossesSimpleVal[end-patience_epochs+1]
        if actual_loss_val > loss_val_prev
            println("Early stopping at epoch: $epoch, because the validation loss is not decreasing after $patience_epochs epochs")
            println("Loss de validación mínimo: ", minimum(lossesSimpleVal), " en la época: ", argmin(lossesSimpleVal))
            break
        end
    end

    # Cleaning garbage memory
    GC.gc()
```

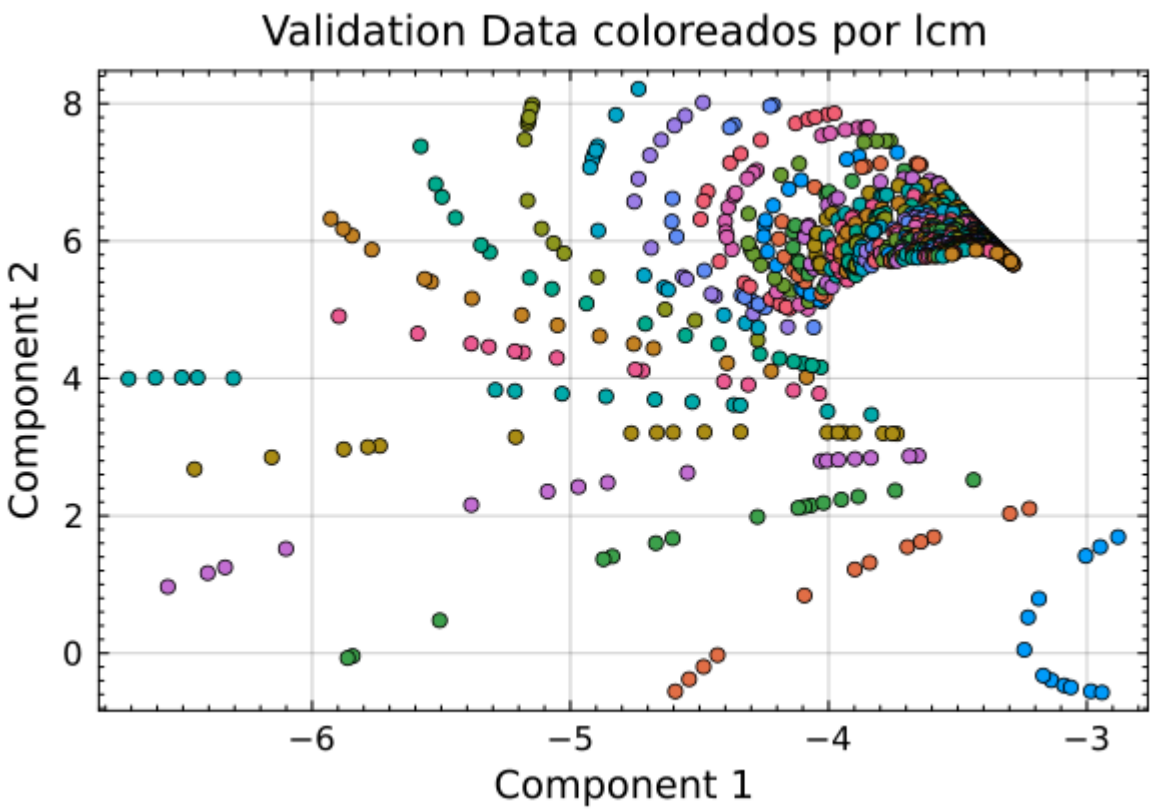


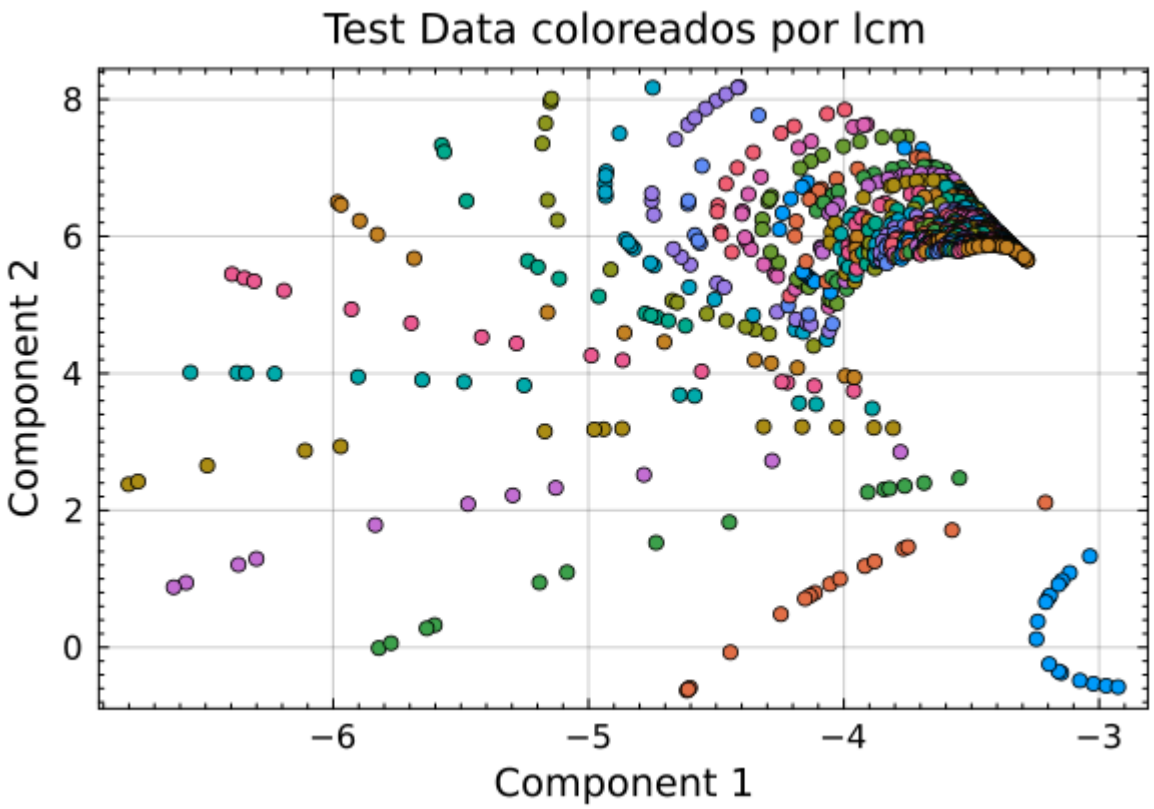
- El loss de validación y de entrenamiento son muy similares por lo que no hay peligro de overfitting

- El error de recosntrucción RMSE del autoencoder simple en el conjunto de señales de test es de: 0.008394564
- Comparado con el error de reconstrucción de PCA (0.006864788124765882) este error del autoencoder simple con activacion lineal es mayor pero del orden.

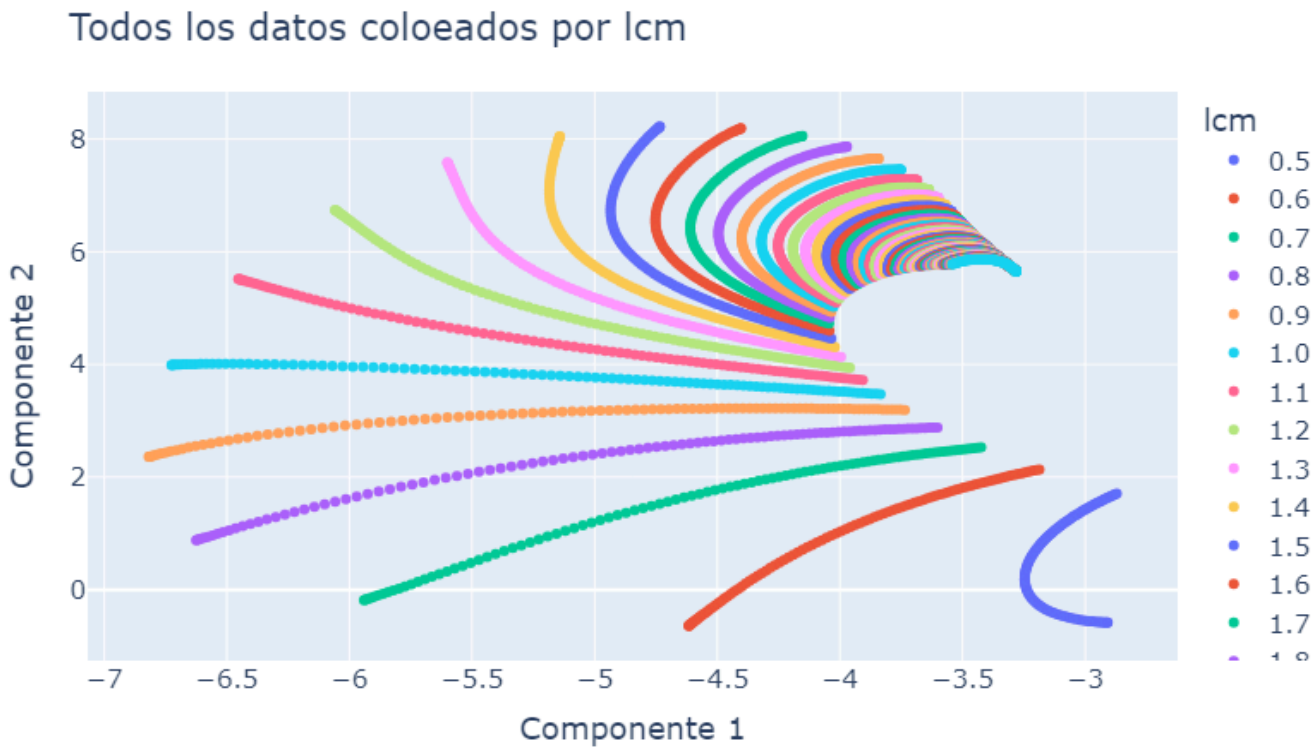
- Grafiquemos el espacio reducido en 2D distinguiendo las señales por l_{cm} como hicimos con PCA.

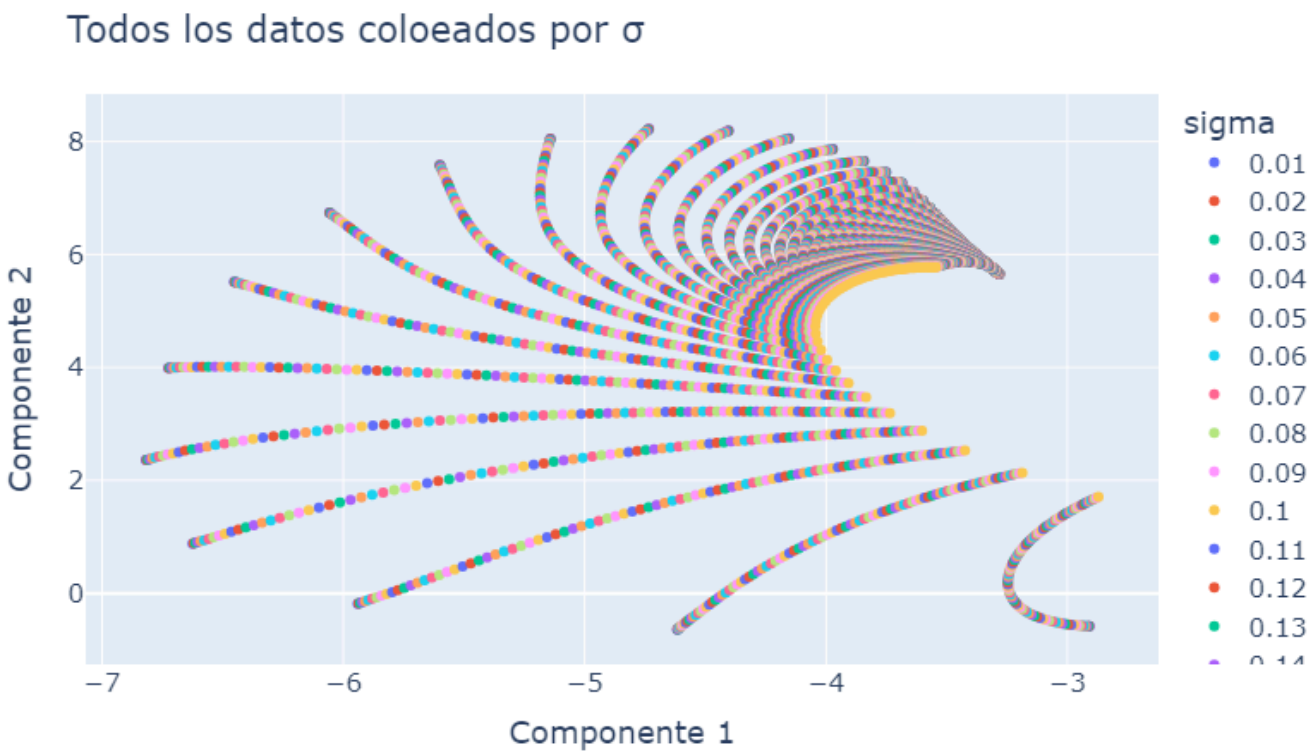

Train Data coloreados por lcm
Component 2
Component 1
8
6
4
2
0
−6
−5
−4
−3


Validation Data coloreados por lcm
Component 2
Component 1
8
6
4
2
0
−6
−5
−4
−3



- Las representaciones de las señales en el espacio latente son muy similares a las de PCA como era de esperarse ya que con activaciones lineales esto es casi equivalente a PCA. Si lo graficamos con plotly podemos tener una mejor visualización.





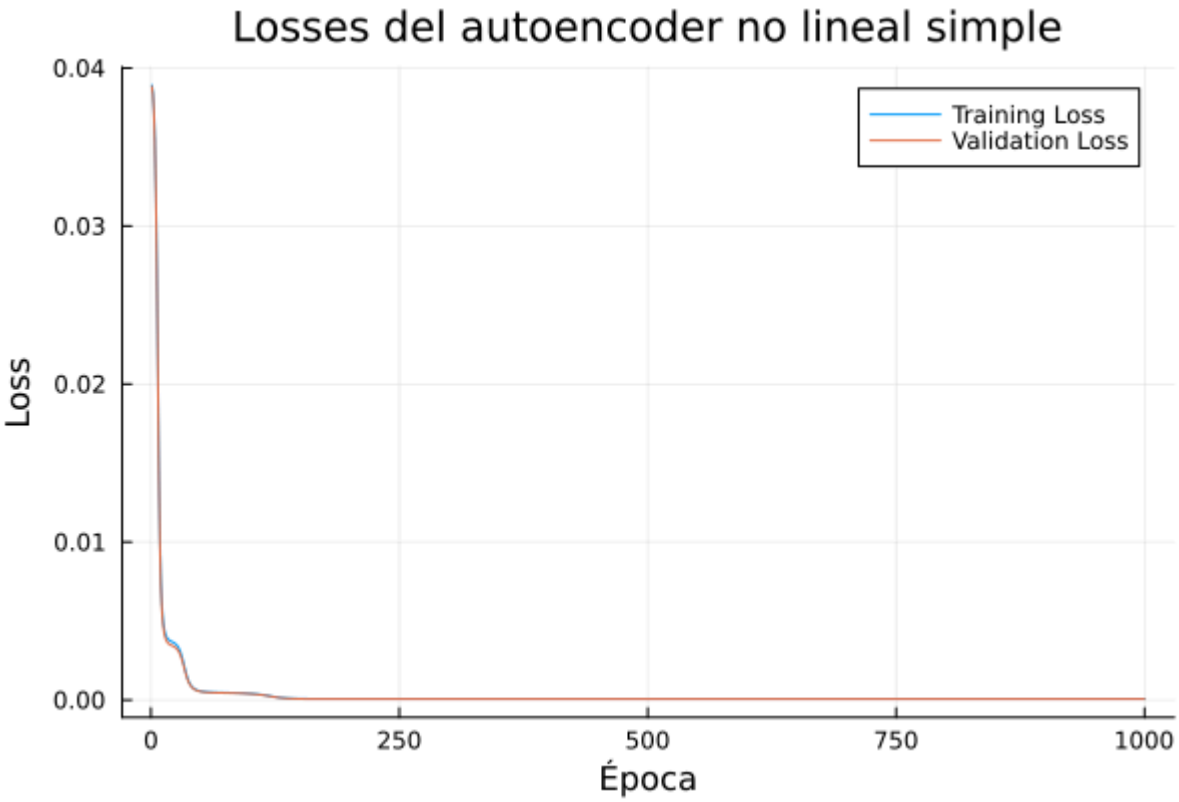
Autoencoder simple con activaciones no lineales

Multiple expressions in one cell.

How would you like to fix it?

- Split this cell into 3 cells, or
- Wrap all code in a *begin ... end* block.

```
# Autoencoder simple
encoderSimpleNoLineal = Chain(Dense(n_times, 3, relu, bias = false))
decoderSimpleNoLineal = Chain(Dense(3, n_times, relu, bias = false))
autoencoderSimpleNoLineal = Chain(encoderSimpleNoLineal, decoderSimpleNoLineal)
```



- Lo mismo pasa con activaciones no lineales

- Calculando el error de recosntrucción RMSE del autoencoder simple en el conjunto de test: 0.008394564
- Comparado con el error de reconstrucción de PCA (0.006864788124765882) este error del autoencoder simple con activacion no lineal es mayor.
- Cuando graficamos la reducción de dimensionalidad queda algo similar al caso anterior con activaciones lineales, esto puede deberse a que la estructura del autoencoder es similar.

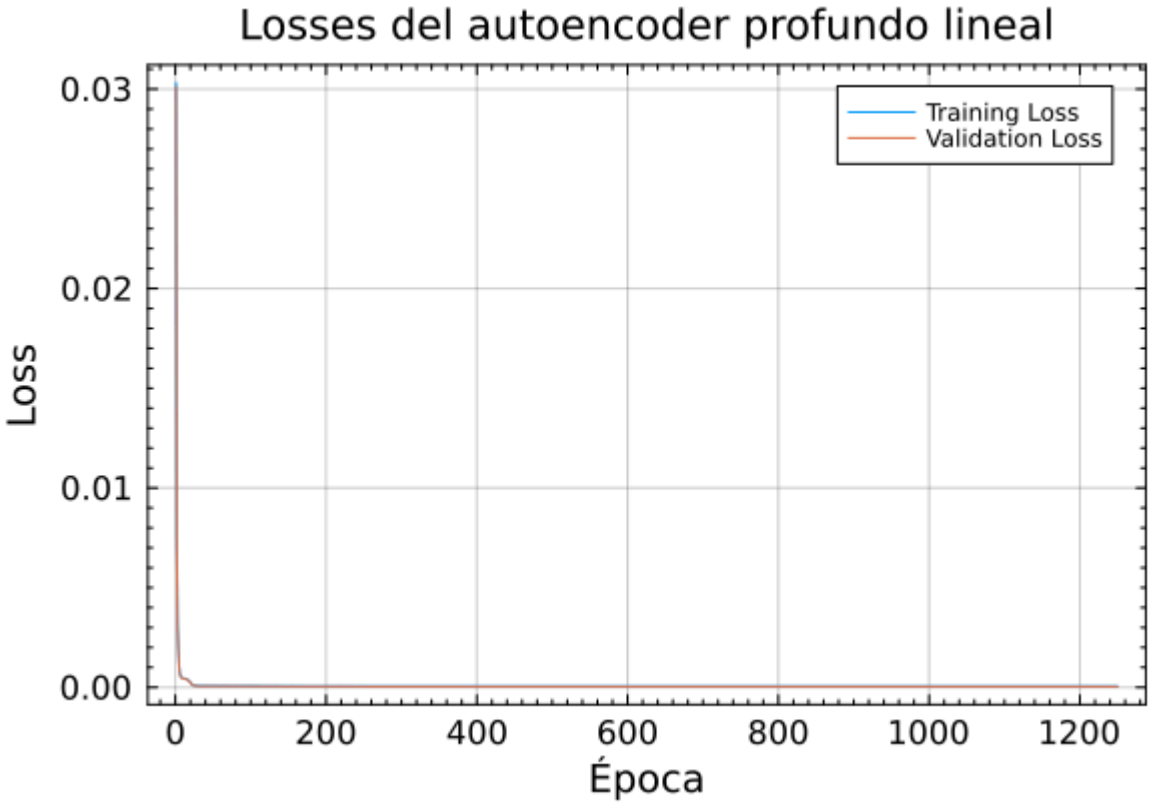
Autoencoder con 4 capas ocultas

Caso con activaciones lineales

- En este caso vamos a ver la representación de las señales en el espacio latente con el autoencoder con 4 capas ocultas y activaciones lineales.
- Vamos a comparar el error de reconstrucción con PCA.

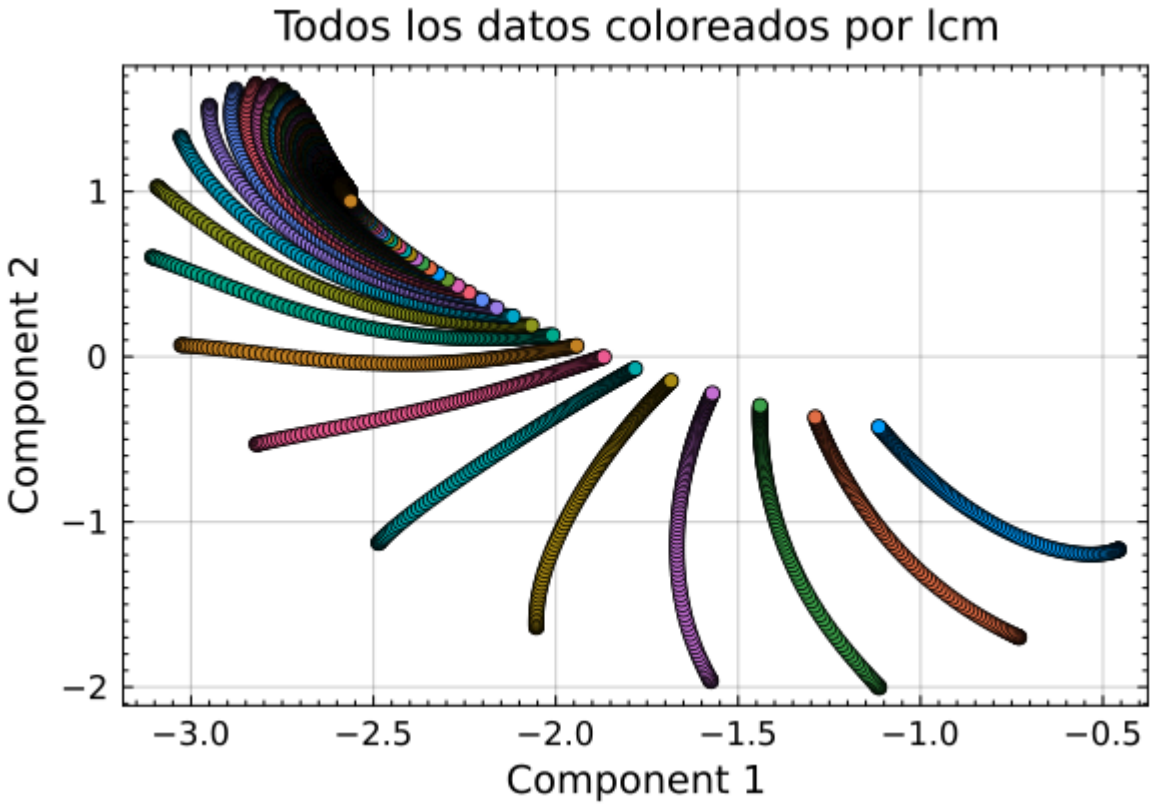
> **Multiple expressions in one cell.**
> How would you like to fix it?
>
> - Split this cell into 3 cells, or
> - Wrap all code in a *begin ... end* block.

```
encoderDeepLineal = Chain(Dense(n_times, 100, identity, bias = false), Dense(100, 50, identity), Dense(50, 3, identity))

decoderDeepLineal = Chain(Dense(3, 50, identity, bias = false), Dense(50, 100, identity), Dense(100, n_times, identity))

autoencoderDeepLineal = Chain(encoderDeepLineal, decoderDeepLineal)
```



- Error de recosntrucción RMSE del autoencoder simple en el conjunto de test: 0.006836749
- Comparado con el error de reconstrucción de PCA (0.006864788124765882) este error del autoencoder con 4 capas ocultas y activacion lineal es casi igual.

Si vemos la recosntrucción de los datos:

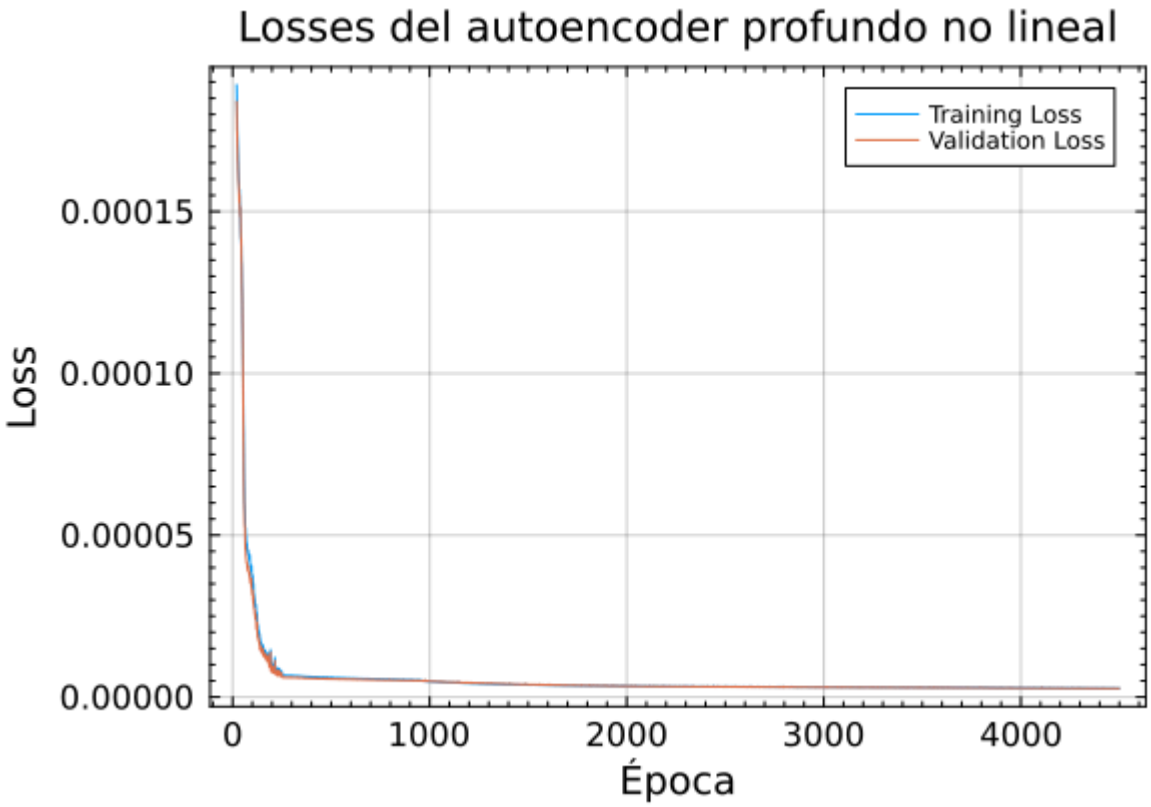


También es similar al PCA original

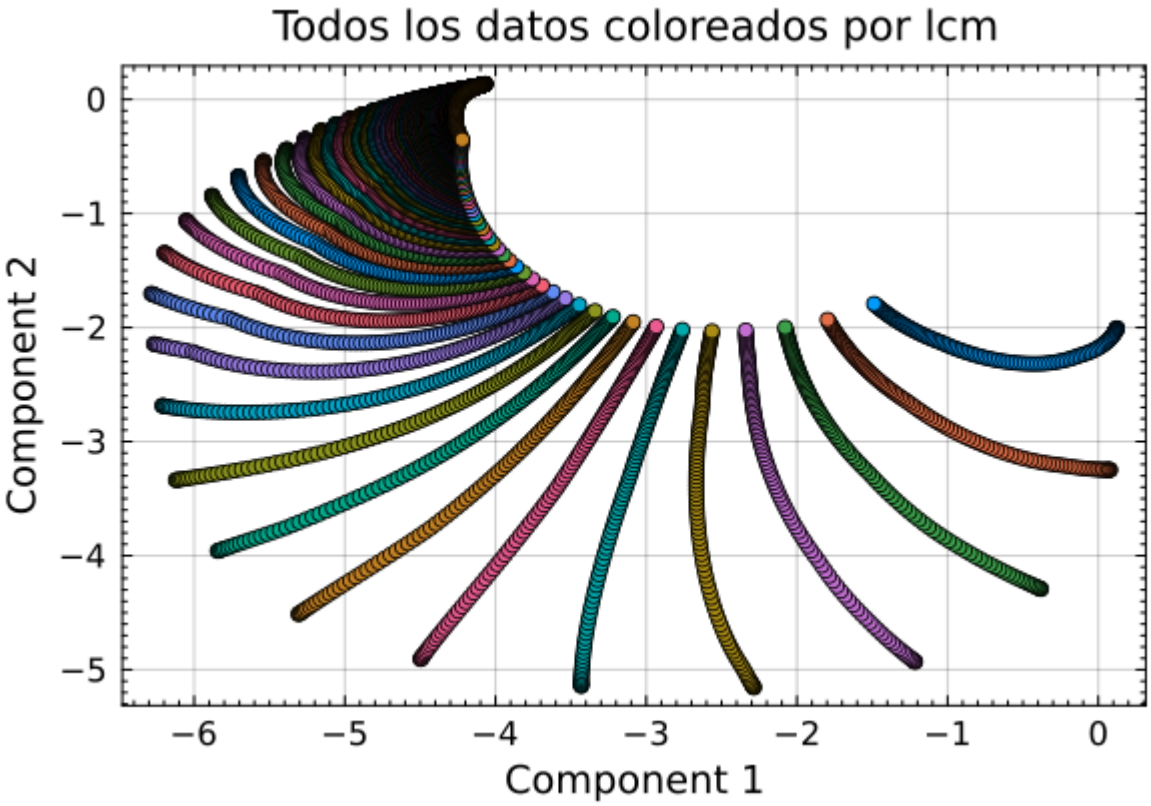
Caso con activaciones no lineales

> **Multiple expressions in one cell.**
> How would you like to fix it?
>
> - Split this cell into 3 cells, or
> - Wrap all code in a *begin ... end* block.

```
encoderDeepNoLineal = Chain(Dense(n_times, 100, bias = false), Dense(100, 50, relu), Dense(50, 3))
decoderDeepNoLineal = Chain(Dense(3, 50, bias = false), Dense(50, 100, relu), Dense(100, n_times))
autoencoderDeepNoLineal = Chain(encoderDeepNoLineal, decoderDeepNoLineal)
```



- Error de recosntrucción RMSE del autoencoder no lineal profundo en el conjunto de test: 0.0016314732
- Comparado con el error de reconstrucción de PCA (0.006864788124765882) este error del autoencoder con 4 capas ocultas y activacion no lineal es menor pero también del orden.

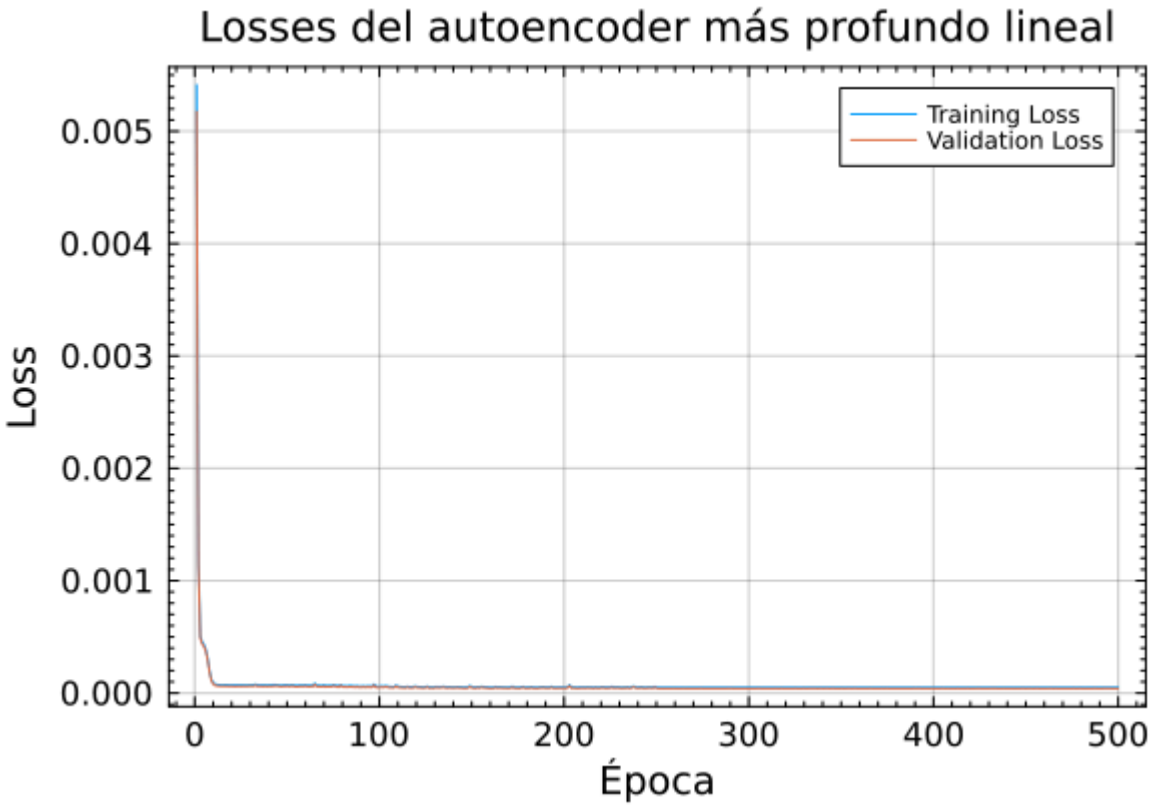


Los cambios son que con las activaciones no lineales ahora los datos parecen estar menos amontonados para valores de lcm y sigma altos. sin embargo el error de reconstrucción es similar al

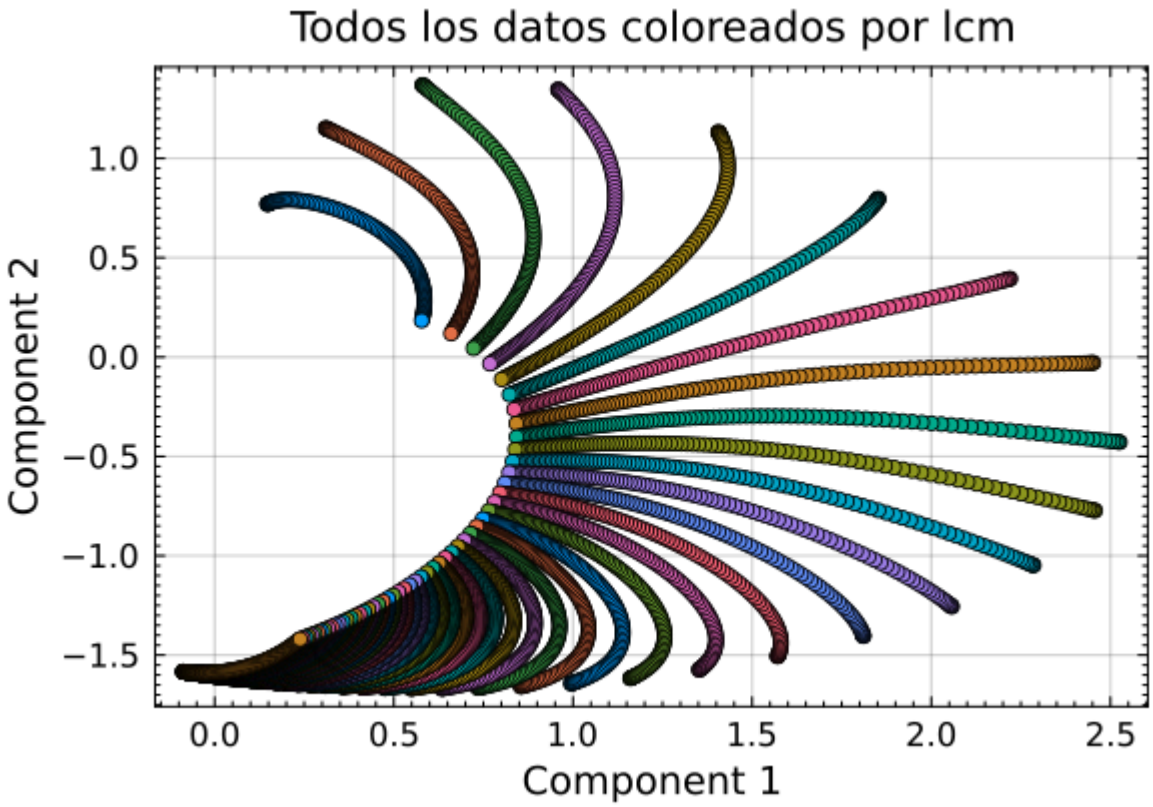
de PCA

Autoencoder con 6 capas ocultas

- En este caso vamos a ver la representación de las señales en el espacio latente con el autoencoder con 6 capas ocultas y activaciones lineales.



- Error de recosntrucción RMSE del autoencoder simple en el conjunto de test: 0.006838203
- Comparado con el error de reconstrucción de PCA (0.006864788124765882) este error del autoencoder con 6 capas ocultas y activacion lineal es similar.
- El error aumentó al agregar capas siendo un autoencoder con activaciones lineales.



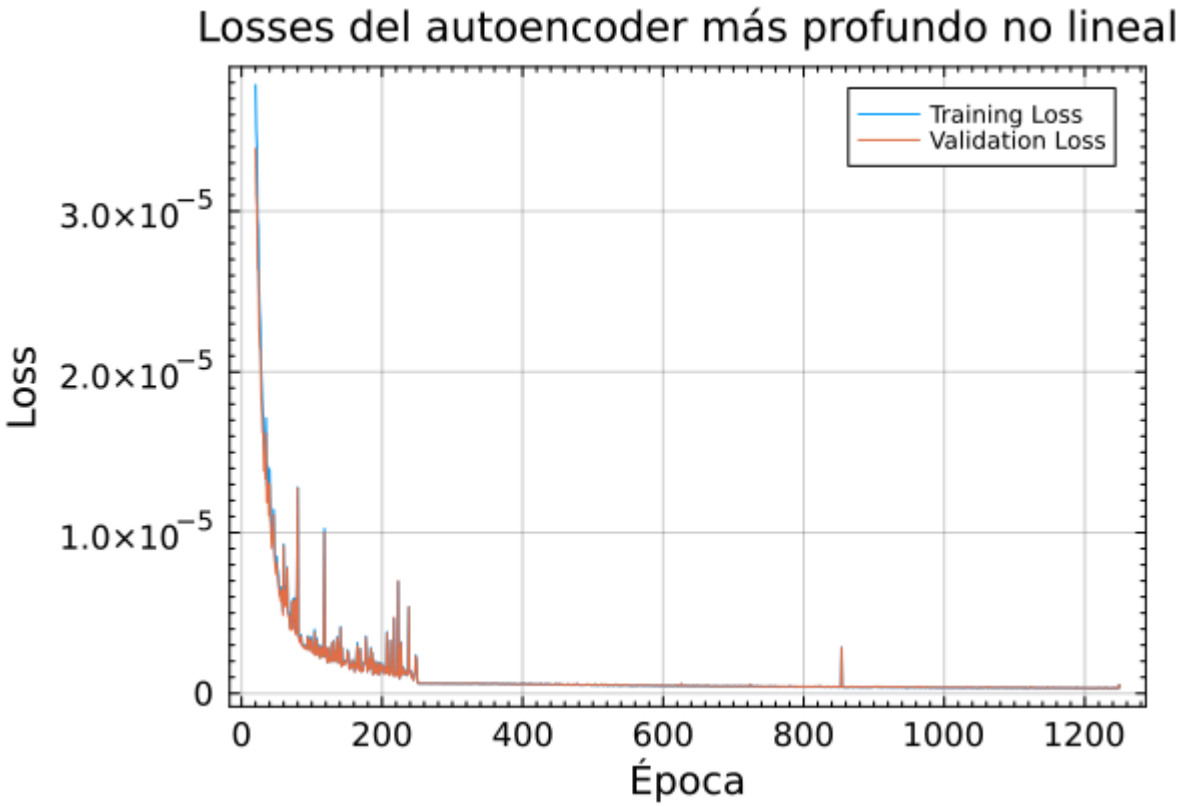
Ahora la representación está cambiando en el sentido que parece rotarse con respecto a las otras componentes, esto tiene sentido en el hecho de que los autoencoders no tienen necesariamente las mismas restricciones de ortogonalidad que impone el PCA, lo que genera diferencias en las representaciones.

En el caso del error este aumentó con respecto al anteriro autoencoder lineal o bueno es casi igual, por lo que probablente sea el límite y seguir aumentando las capas lleve al overfitting.

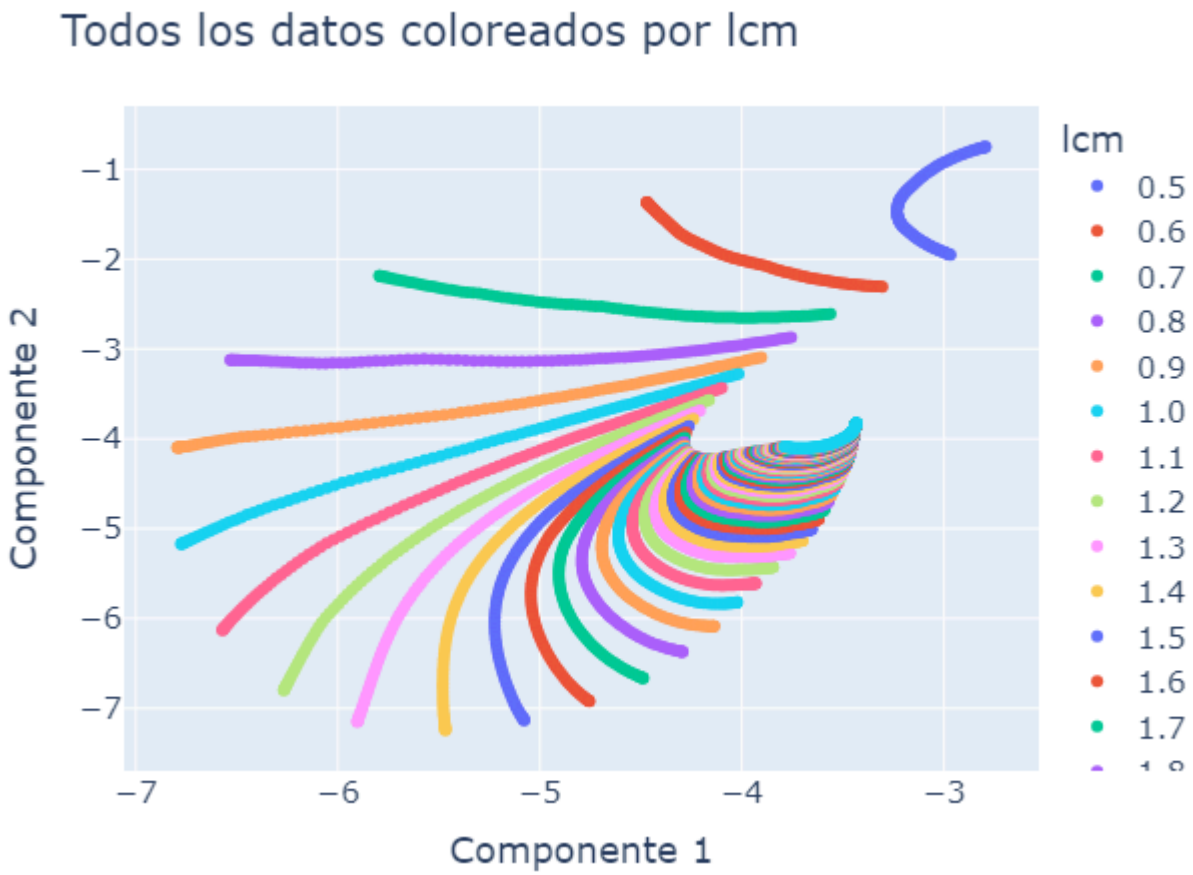
Caso con activaciones no lineales

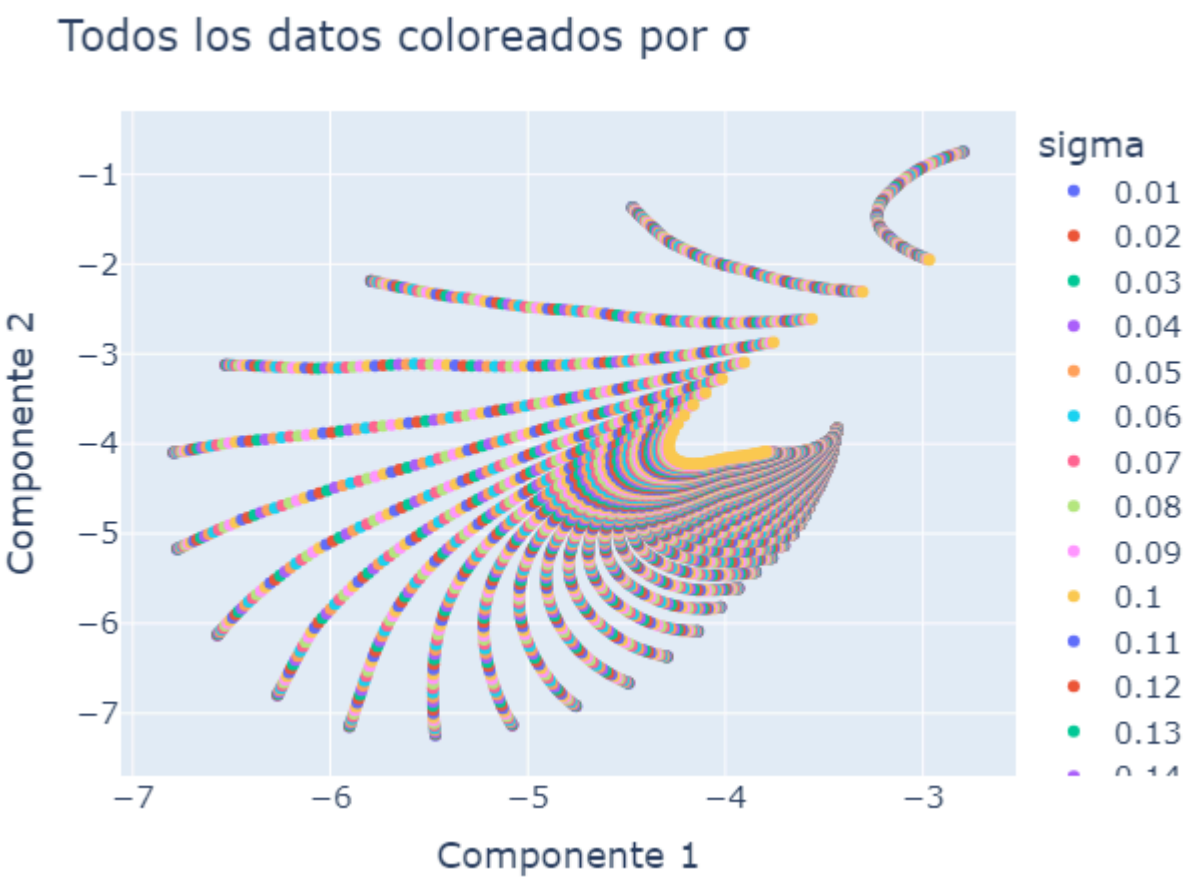
> **Multiple expressions in one cell.**
> How would you like to fix it?
>
> - Split this cell into 3 cells, or
> - Wrap all code in a *begin ... end* block.

```
encoderDeeperNoLineal = Chain(Dense(n_times, 500, bias = false), Dense(500, 100, relu), Dense(100, 50, relu), Dense(50, 3))
decoderDeeperNoLineal = Chain(Dense(3, 50, bias = false), Dense(50, 100, relu), Dense(100, 500, relu), Dense(500, n_times))
autoencoderDeeperNoLineal = Chain(encoderDeeperNoLineal, decoderDeeperNoLineal)
```



- Error de recosntrucción RMSE del autoencoder simple en el conjunto de test: 0.00056004466
- Comparado con el error de reconstrucción de PCA (0.006864788124765882) este error del autoencoder con 6 capas ocultas y activacion no lineal es mucho menor, incluso menor que el autoencoder no lineal con 4 capas ocultas.





Ahora la represenación si camibia con respecto a PCA y la transformación que se aprende puede involucrar cambios no lineales en el espacio latente que no tienen una interpretación tan directa como en el PCA