Actividad_Autoencodes

November 22, 2023

0.1 Actividad: Autoencodes

Implementen un autoencoder para la base de datos de "Emojis". Para ello, sigan los siguientes pasos:

1. Dividan aleatoriamente su conjunto de datos de tal manera que el 80% de los datos sean para entrenamiento y un 20% para prueba. Procuren que en la división las clases mantengan la misma proporción tanto en los datos de entrenamiento como en los de prueba respecto a las proporciones.

[ ]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from collections import Counter


data = np.loadtxt("emojis.txt")
x = data[:,1:]
y = data[:,0]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.
↪2,stratify=y)
```

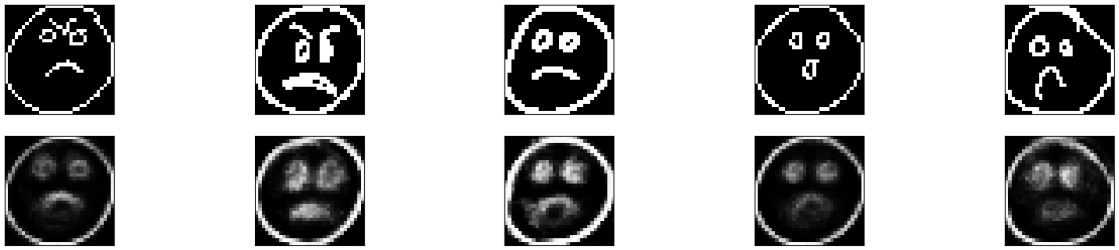
2. Sigan los pasos del tutorial "Building autoencoders with Keras" para ajustar un autoencoder de una capa oculta para los datos de Emojis utilizando los datos de entrenamiento.

[ ]:
```
import keras
from keras import layers
import matplotlib.pyplot as plt

encoding_dim = 32

input_img = keras.Input(shape=(x_train.shape[1]))
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
decoded = layers.Dense(x_train.shape[1], activation='sigmoid')(encoded)

autoencoder = keras.Model(input_img, decoded)
encoder = keras.Model(input_img, encoded)

encoded_input = keras.Input(shape=(encoding_dim,))
```

```
decoder_layer = autoencoder.layers[-1]
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))


# 3. Utilicen algunas imágenes del conjunto de prueba para ver la salida de la␣
 ↪capa intermedia
# y de la capa de decodificación.

encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)

n = 5
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

#4. Evalúen el error cuadrático medio entre las imágenes de prueba
# y su correspondiente salida.

from sklearn.metrics import mean_squared_error

mse_test = mean_squared_error(x_test, decoded_imgs)

print(f'MSE | Datos de prueba: {mse_test}')
```

```
Epoch 1/50
8/8 [==============================] - 1s 45ms/step - loss: 0.6772 - val_loss:
0.6388
```

```
Epoch 50/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2557 - val_loss:
0.2617
16/16 [==============================] - 0s 1ms/step
16/16 [==============================] - 0s 1ms/step
MSE | Datos de prueba: 0.08176597702897656
```



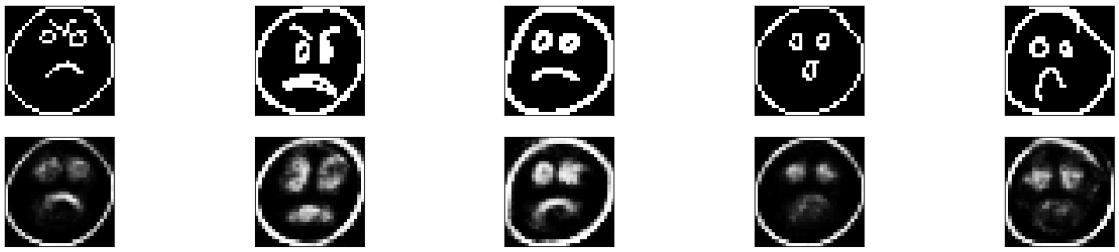
5. Agreguen un factor de regularización a la capa intermedia y repitan los pasos 3 y 4 con este nuevo modelo.

[ ]:
```python
import keras
from keras import layers
import matplotlib.pyplot as plt
from keras import regularizers


encoding_dim = 32

input_img = keras.Input(shape=(x_train.shape[1]))

# Agregamos un factor de regularización a nuestra capa intermedia
encoded = layers.Dense(encoding_dim, activation='relu',
                activity_regularizer=regularizers.l1(10e-5))(input_img)
decoded = layers.Dense(x_train.shape[1], activation='sigmoid')(encoded)

autoencoder = keras.Model(input_img, decoded)
encoder = keras.Model(input_img, encoded)

encoded_input = keras.Input(shape=(encoding_dim,))
decoder_layer = autoencoder.layers[-1]
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

autoencoder.fit(x_train, x_train,
                epochs=100,
                batch_size=256,
```

```
                shuffle=True,
                validation_data=(x_test, x_test))


# 3. Utilicen algunas imágenes del conjunto de prueba para ver la salida de la␣
 ↪capa intermedia
# y de la capa de decodificación.

encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)

n = 5
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

#4. Evalúen el error cuadrático medio entre las imágenes de prueba
# y su correspondiente salida.

from sklearn.metrics import mean_squared_error

mse_test = mean_squared_error(x_test, decoded_imgs)

print(f'MSE | Datos de prueba: {mse_test}')
```

```
Epoch 1/100
8/8 [==============================] - 1s 29ms/step - loss: 0.6821 - val_loss:
0.6510
Epoch 2/100
8/8 [==============================] - 0s 16ms/step - loss: 0.6063 - val_loss:
0.5381
Epoch 3/100
8/8 [==============================] - 0s 16ms/step - loss: 0.5079 - val_loss:
0.4704
Epoch 4/100
8/8 [==============================] - 0s 16ms/step - loss: 0.4637 - val_loss:
```

```
0.2466
16/16 [==============================] - 0s 1ms/step
16/16 [==============================] - 0s 784us/step
MSE | Datos de prueba: 0.07330598304472671
```



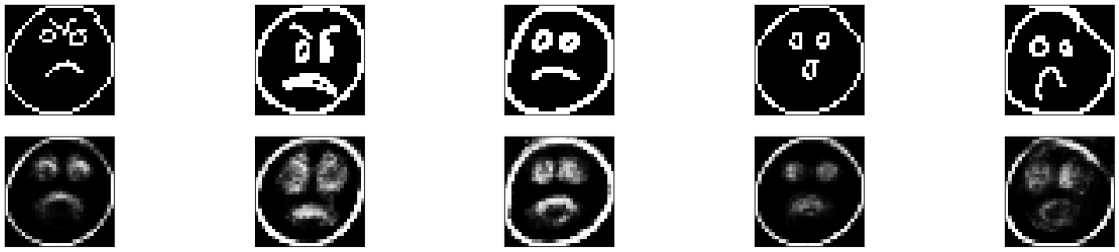
6. Implemente un autoencoder profundo agregando más capas internas, y evalúen este nuevo modelo tal como se hizo en los dos modelos anteriores.

[ ]:
```
encoding_dim = 32

input_img = keras.Input(shape=(x_train.shape[1],))
encoded = layers.Dense(128, activation='relu')(input_img)
encoded = layers.Dense(64, activation='relu')(encoded)
encoded = layers.Dense(encoding_dim, activation='relu')(encoded)

decoded = layers.Dense(64, activation='relu')(encoded)
decoded = layers.Dense(128, activation='relu')(decoded)
decoded = layers.Dense(x_train.shape[1], activation='sigmoid')(decoded)

autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

autoencoder.fit(x_train, x_train,
                epochs=100,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))

# 3. Utilicen algunas imágenes del conjunto de prueba para ver la salida de la␣
 ↪capa intermedia
# y de la capa de decodificación.

decoded_imgs = autoencoder.predict(x_test)

n = 5
plt.figure(figsize=(20, 4))
for i in range(n):
```

```
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

#4. Evalúen el error cuadrático medio entre las imágenes de prueba
# y su correspondiente salida.

from sklearn.metrics import mean_squared_error

mse_test = mean_squared_error(x_test, decoded_imgs)

print(f'MSE | Datos de prueba: {mse_test}')
```

```
Epoch 1/100
8/8 [==============================] - 1s 33ms/step - loss: 0.6765 - val_loss:
0.6087
Epoch 2/100
8/8 [==============================] - 0s 21ms/step - loss: 0.5372 - val_loss:
0.4682
Epoch 3/100
8/8 [==============================] - 0s 18ms/step - loss: 0.4483 - val_loss:
0.4297
Epoch 4/100
8/8 [==============================] - 0s 21ms/step - loss: 0.4244 - val_loss:
0.4141
Epoch 5/100
8/8 [==============================] - 0s 18ms/step - loss: 0.4115 - val_loss:
0.4041
Epoch 6/100
8/8 [==============================] - 0s 20ms/step - loss: 0.4012 - val_loss:
0.3923
Epoch 7/100
8/8 [==============================] - 0s 21ms/step - loss: 0.3904 - val_loss:
0.3809
Epoch 8/100
8/8 [==============================] - 0s 19ms/step - loss: 0.3803 - val_loss:
0.3742
Epoch 9/100
```

```
8/8 [==============================] - 0s 20ms/step - loss: 0.2208 - val_loss:
0.2510
Epoch 90/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2201 - val_loss:
0.2500
Epoch 91/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2195 - val_loss:
0.2503
Epoch 92/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2191 - val_loss:
0.2503
Epoch 93/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2185 - val_loss:
0.2504
Epoch 94/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2180 - val_loss:
0.2500
Epoch 95/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2174 - val_loss:
0.2499
Epoch 96/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2165 - val_loss:
0.2496
Epoch 97/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2160 - val_loss:
0.2494
Epoch 98/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2155 - val_loss:
0.2492
Epoch 99/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2149 - val_loss:
0.2486
Epoch 100/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2142 - val_loss:
0.2488
16/16 [==============================] - 0s 2ms/step
MSE | Datos de prueba: 0.07815937762367585
```



7. Agreguen ruido binario a algunas imágenes de prueba (es decir, prendan o apaguen algunos pixeles de manera aleatoria), y verifiquen la salida del autoencoder profundo. ¿El modelo es capaz de eliminar ruido en las imágenes de entrada?

[ ]:
```
x_train = np.reshape(x_train, (len(x_train), 32, 32, 1))
x_test = np.reshape(x_test, (len(x_test), 32, 32, 1))

noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
 ↪size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
 ↪size=x_test.shape)

x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)

n = 5
plt.figure(figsize=(20, 2))
for i in range(1, n + 1):
    ax = plt.subplot(1, n, i)
    plt.imshow(x_test_noisy[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

[ ]:
```
from keras.callbacks import TensorBoard


input_img = keras.Input(shape=(32, 32, 1))

x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
```

```
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

autoencoder.fit(x_train_noisy, x_train,
                epochs=100,
                batch_size=128,
                shuffle=True,
                validation_data=(x_test_noisy, x_test),
                callbacks=[TensorBoard(log_dir='/tmp/tb', histogram_freq=0,␣
 ↪write_graph=False)])
```

```
Epoch 1/100
16/16 [==============================] - 5s 239ms/step - loss: 0.5537 -
val_loss: 0.5131
Epoch 2/100
16/16 [==============================] - 4s 227ms/step - loss: 0.5082 -
val_loss: 0.4967
Epoch 3/100
16/16 [==============================] - 4s 241ms/step - loss: 0.4976 -
val_loss: 0.4848
Epoch 4/100
16/16 [==============================] - 4s 263ms/step - loss: 0.4840 -
val_loss: 0.4649
Epoch 5/100
16/16 [==============================] - 4s 265ms/step - loss: 0.4557 -
val_loss: 0.4291
Epoch 6/100
16/16 [==============================] - 4s 276ms/step - loss: 0.4150 -
val_loss: 0.3865
Epoch 7/100
16/16 [==============================] - 4s 229ms/step - loss: 0.3742 -
val_loss: 0.3520
Epoch 8/100
16/16 [==============================] - 4s 249ms/step - loss: 0.3438 -
val_loss: 0.3273
Epoch 9/100
16/16 [==============================] - 4s 264ms/step - loss: 0.3221 -
val_loss: 0.3106
Epoch 10/100
16/16 [==============================] - 4s 245ms/step - loss: 0.3052 -
val_loss: 0.2966
Epoch 11/100
16/16 [==============================] - 4s 244ms/step - loss: 0.2923 -
val_loss: 0.2843
Epoch 12/100
16/16 [==============================] - 5s 285ms/step - loss: 0.2811 -
```

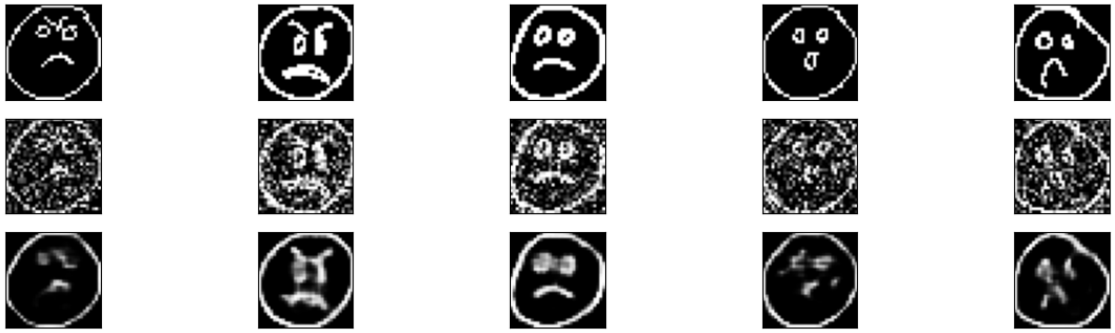
```
val_loss: 0.1726
Epoch 93/100
16/16 [==============================] - 4s 225ms/step - loss: 0.1661 -
val_loss: 0.1703
Epoch 94/100
16/16 [==============================] - 4s 228ms/step - loss: 0.1656 -
val_loss: 0.1705
Epoch 95/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1655 -
val_loss: 0.1702
Epoch 96/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1652 -
val_loss: 0.1700
Epoch 97/100
16/16 [==============================] - 4s 225ms/step - loss: 0.1657 -
val_loss: 0.1715
Epoch 98/100
16/16 [==============================] - 4s 227ms/step - loss: 0.1654 -
val_loss: 0.1693
Epoch 99/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1643 -
val_loss: 0.1688
Epoch 100/100
16/16 [==============================] - 4s 240ms/step - loss: 0.1643 -
val_loss: 0.1720
```

[ ]:
```
<keras.src.callbacks.History at 0x26625c0aaf0>
```

[ ]:
```python
decoded_imgs = autoencoder.predict(x_test_noisy)

n = 5  # número de imágenes a mostrar
plt.figure(figsize=(15, 4))
for i in range(n):
    # Muestra la imagen original
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(x_test[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Muestra la imagen ruidosa
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(x_test_noisy[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
```

```
    # Muestra la imagen reconstruida
    ax = plt.subplot(3, n, i + 1 + 2*n)
    plt.imshow(decoded_imgs[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()
```

```
16/16 [==============================] - 0s 16ms/step
```



¿El modelo es capaz de eliminar ruido en las imágenes de entrada?

Si, el modelo es capaz de eliminar el ruido de las imágenes, esto se debe a que gracias a nuestro modelo es capaz de aprender a extraer las caracteristicas esenciales de las imágenes originales y a reconstruirlas a una imagen más limpia en base a ello.

Cabe mencionar que nuestro modelo usa capas convolucionales, que son capaces de extraer características visuales de alto nivel de las imágenes, y que son más eficientes y robustas que las capas densas. Las capas convolucionales también preservan la estructura espacial de las imágenes, lo que facilita la reconstrucción.