Deep Learning.

Proyecto 1 - Autoencoders.

Nombre: Sergio Daniel Dueñas Godinez.

Expediente : 739300.

Profesor: Iván Reyes Amezcua.

Link Github: https://github.com/SergioDuenass/Camus-GPT

Objetivos:

```
Comprender los principios fundamentales de los autoencoders y su aplicación en deep learn
Implementar un autoencoder básico y variacionales para una tarea específica, como reducci
Analizar el rendimiento y las características de las representaciones aprendidas por los
```

Descripción

```
Deberán seleccionar un conjunto de datos adecuado para su proyecto, que puede ser de imág
Implementar un autoencoder, como un variacional (VAE) o un autoencoder convolucional, dep
El proyecto incluirá una fase de experimentación donde los deberán entrenar, ajustar y ev
Presentar sus resultados a través de un informe escrito y una presentación, discutir la i
```

Los datos fueron recopliados de los libros de Camus; 'El Extranjero' y 'La Plaga'

```
# Librerias a utilizar
import tensorflow as tf
from tensorflow.keras.models import Model
```

```
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.losses import binary_crossentropy, mean_squared_error
from tensorflow.keras.layers import Input, Dense, Flatten, Reshape, Conv1D, Conv1

import numpy as np
import spacy
import re


from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

Tokenización, embeddings y limpieza de corpus

```
def clean_corpus_from_file(file_path):
    cleaned_corpus = []

    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            # Leer el contenido del archivo
            corpus = file.readlines()

            for text in corpus:
                # Convertir a minúsculas
                text = text.lower()

                # Eliminar caracteres no alfabéticos y números
                text = re.sub(r'[^a-z\s]', '', text)

                # Eliminar espacios en blanco adicionales
                text = ' '.join(text.split())

                cleaned_corpus.append(text)

    except Exception as e:
        print(f"Error al leer el archivo: {e}")

    return cleaned_corpus



def clean_corpus(corpus):
    cleaned_corpus = []

    for text in corpus:
```

```
    for text in corpus:
        # Convertir a minúsculas
        text = text.lower()

        # Eliminar caracteres no alfabéticos y números
        text = re.sub(r'[^a-z\s]', '', text)

        # Eliminar espacios en blanco adicionales
        text = ' '.join(text.split())

        cleaned_corpus.append(text)

    return cleaned_corpus

# Utilizo spacy para la tokenización
nlp = spacy.load("en_core_web_sm")

def tokenize_corpus(corpus):
    tokenized_corpus = []

    for text in corpus:
        # Tokenizar el texto usando spaCy
        tokens = [token.text for token in nlp(text)]
        tokenized_corpus.append(tokens)

    return tokenized_corpus

corpus_file_path = "/content/drive/MyDrive/ProyectoCamus/data/corpus.txt"
cleaned_corpus = clean_corpus_from_file(corpus_file_path)

tokenized_corpus = tokenize_corpus(cleaned_corpus)


# Construir un vocabulario
vocabulario = {token: idx for idx, token in enumerate(set(token for sublist in tc
vocab_size = len(vocabulario)

# Convertir tokens a índices
corpus_indices = [[vocabulario[token] for token in secuencia] for secuencia in tc

# Padding de las secuencias
corpus_padded = pad_sequences(corpus_indices, padding='post', truncating='post')

# Crear el modelo de embedding
embedding_dim = 300  # ajusta según tus necesidades
embedding_model = Embedding(input_dim=vocab_size, output_dim=embedding_dim)

# Obtener vectores de embedding
embedded_sequence = embedding_model(corpus_padded)

# Ver la salida
```

```
print("Corpus Padded Shape:", corpus_padded.shape)
print("Embedded Sequence Shape:", embedded_sequence.shape)
```

```
Corpus Padded Shape: (8289, 106)
Embedded Sequence Shape: (8289, 106, 300)
```

VAE

```
embedded_sequence.shape
```

```
TensorShape([8289, 106, 300])
```

```
input_shape = embedded_sequence.shape[1:]
batch_size = 128
latent_dim = 64
epochs = 30
```

Double-click (or enter) to edit

```
def sampling(args):
  z_mean, z_log_var = args

  dim = K.int_shape(z_mean)[1]

  # TODO: check dimensions
  epsilon = K.random_normal(shape = (K.shape(z_mean)[0], dim))

  return z_mean + K.exp(0.5 * z_log_var) * epsilon

# Ajustar las dimensiones de las secuencias de entrada
target_length_input = 106
corpus_padded_input = pad_sequences(corpus_indices, maxlen=target_length_input, p

# Convertir a embeddings
embedded_sequence_input = embedding_model(corpus_padded_input)

# Ajustar las dimensiones de las secuencias de salida
target_length_output = 106
corpus_padded_output = pad_sequences(corpus_indices, maxlen=target_length_output,


# Convertir a embeddings
embedded_sequence_output = embedding_model(corpus_padded_output)



embedded_sequence_input.shape, embedded_sequence_output.shape
```

```
embedded_sequence_input.shape, embedded_sequence_output.shape
```

```
(TensorShape([8289, 106, 300]), TensorShape([8289, 106, 300]))
```

```
outputs_decoder
```

```
<KerasTensor: shape=(None, 108, 300) dtype=float32 (created by layer
'conv1d_transpose_2')>
```

```
# Ajustar el modelo Encoder
input_shape_encoder = (target_length_output, embedding_dim)  # Ajusta según tus r
latent_dim = 32  # Ajusta según tus necesidades

inputs_encoder = Input(shape=input_shape_encoder, name="encoder_input")
x_encoder = Conv1D(32, 3, activation="relu", strides=2, padding="same")(inputs_er
x_encoder = Conv1D(64, 3, activation="relu", strides=2, padding="same")(x_encoder

shape_before_flat_encoder = K.int_shape(x_encoder)
x_encoder = Flatten()(x_encoder)
x_encoder = Dense(256, activation="relu")(x_encoder)

z_mean_encoder = Dense(latent_dim, name='z_mean')(x_encoder)
z_log_var_encoder = Dense(latent_dim, name='z_log_var')(x_encoder)

encoder = Model(inputs_encoder, [z_mean_encoder, z_log_var_encoder], name='encode

#
target_length_output = 106
embedding_dim = 300

# Definir el modelo Decoder
latent_inputs_decoder = Input(shape=(latent_dim,), name='z_sampling')
x_decoder = Dense(np.prod(shape_before_flat_encoder[1:]), activation="relu")(late
x_decoder = Reshape(shape_before_flat_encoder[1:])(x_decoder)

# Usar Conv1DTranspose con padding='same' para ajustar la longitud de salida
x_decoder = Conv1DTranspose(64, 3, activation="relu", strides=2, padding="same")(
x_decoder = Conv1DTranspose(32, 3, activation="relu", strides=2, padding="same")(

# Ajustar manualmente la longitud de la salida a target_length_output
outputs_decoder = Conv1DTranspose(embedding_dim, 3, activation="sigmoid", padding
outputs_decoder = outputs_decoder[:, :target_length_output, :]

# Definir el modelo Decoder
decoder = Model(latent_inputs_decoder, outputs_decoder, name='decoder')

# VAE
outputs_vae = decoder(encoder(inputs_encoder)[0])
vae = Model(inputs_encoder, outputs_vae, name='vae')

# Compilar el modelo con la función de pérdida personalizada
```

```
vae.compile(optimizer='adam', loss=tf.keras.losses.MeanSquaredError())

# Resumen del modelo
vae.summary()
```

```
Model: "vae"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder_input (InputLayer)  [(None, 106, 300)]        0

 encoder (Functional)        [(None, 32),              494112
                              (None, 32)]

 decoder (Functional)        (None, 106, 300)          104652

=================================================================
Total params: 598764 (2.28 MB)
Trainable params: 598764 (2.28 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Entrenar el modelo
vae.fit(embedded_sequence_input, embedded_sequence_output, epochs=epochs, batch_s
```

```
Epoch 1/30
WARNING:tensorflow:Gradients do not exist for variables ['z_log_var/kernel:0'
WARNING:tensorflow:Gradients do not exist for variables ['z_log_var/kernel:0'
WARNING:tensorflow:Gradients do not exist for variables ['z_log_var/kernel:0'
WARNING:tensorflow:Gradients do not exist for variables ['z_log_var/kernel:0'
65/65 [==============================] - 21s 287ms/step - loss: 0.0712
Epoch 2/30
65/65 [==============================] - 21s 320ms/step - loss: 8.7673e-04
Epoch 3/30
65/65 [==============================] - 18s 269ms/step - loss: 8.7673e-04
Epoch 4/30
65/65 [==============================] - 18s 275ms/step - loss: 8.7673e-04
Epoch 5/30
65/65 [==============================] - 16s 252ms/step - loss: 8.7673e-04
Epoch 6/30
65/65 [==============================] - 17s 257ms/step - loss: 8.7673e-04
Epoch 7/30
65/65 [==============================] - 18s 272ms/step - loss: 8.7673e-04
Epoch 8/30
65/65 [==============================] - 17s 266ms/step - loss: 8.7673e-04
Epoch 9/30
65/65 [==============================] - 17s 260ms/step - loss: 8.7673e-04
Epoch 10/30
65/65 [==============================] - 17s 258ms/step - loss: 8.7673e-04
Epoch 11/30
65/65 [==============================] - 18s 271ms/step - loss: 8.7673e-04
Epoch 12/30
65/65 [==============================] - 19s 283ms/step - loss: 8.7673e-04
Epoch 13/30
```

```
65/65 [==============================] - 17s 261ms/step - loss: 8.7673e-04
Epoch 14/30
65/65 [==============================] - 17s 260ms/step - loss: 8.7673e-04
Epoch 15/30
65/65 [==============================] - 17s 260ms/step - loss: 8.7673e-04
Epoch 16/30
65/65 [==============================] - 18s 276ms/step - loss: 8.7673e-04
Epoch 17/30
65/65 [==============================] - 18s 276ms/step - loss: 8.7673e-04
Epoch 18/30
65/65 [==============================] - 17s 262ms/step - loss: 8.7673e-04
Epoch 19/30
65/65 [==============================] - 17s 257ms/step - loss: 8.7673e-04
Epoch 20/30
65/65 [==============================] - 18s 271ms/step - loss: 8.7673e-04
Epoch 21/30
65/65 [==============================] - 19s 284ms/step - loss: 8.7673e-04
Epoch 22/30
65/65 [==============================] - 17s 265ms/step - loss: 8.7673e-04
Epoch 23/30
65/65 [==============================] - 17s 259ms/step - loss: 8.7673e-04
Epoch 24/30
65/65 [==============================] - 17s 256ms/step - loss: 8.7673e-04
Epoch 25/30
65/65 [==============================] - 18s 272ms/step - loss: 8.7673e-04
Epoch 26/30
65/65 [==============================] - 17s 265ms/step - loss: 8.7673e-04
Epoch 27/30
65/65 [==============================] - 17s 258ms/step - loss: 8.7673e-04
```

```
# Obtener z_mean, z_log_var
z_mean_batch, z_log_var_batch = encoder.predict(embedded_sequence_input)

# Utilizar la función de muestreo para generar muestras de la distribución latent
latent_samples = sampling([z_mean_batch, z_log_var_batch])

# Decodificar las muestras generadas para obtener nuevos embeddings
decoded_images = decoder.predict(latent_samples)
```

```
260/260 [==============================] - 2s 7ms/step
260/260 [==============================] - 5s 20ms/step
```

Cálamos el modelito

```
# Crear el vocabulario inverso
inv_vocabulario = {idx: palabra for palabra, idx in vocabulario.items()}

# Convertir los índices de embedding a palabras usando el vocabulario inverso
palabras_recuperadas = []
```

```
for idx_mas_cercano in decoded_images[0]:
    # Verificar si el índice está en el rango del vocabulario
    if np.all(idx_mas_cercano < len(inv_vocabulario)):
        # Si idx_mas_cercano es un array, obtener las palabras correspondientes a
        if isinstance(idx_mas_cercano, np.ndarray):
            palabras_recuperadas.extend([inv_vocabulario[idx] for idx in idx_mas_
        else:
            # Si es un escalar, obtener la palabra correspondiente al índice
            palabra_recuperada = inv_vocabulario[idx_mas_cercano]
            palabras_recuperadas.append(palabra_recuperada)
    else:
        print("Índice más cercano fuera de rango del vocabulario")

# Unir las palabras para obtener la oración original
oracion_original = ' '.join(palabras_recuperadas)

# Imprimir la oración original
print("Oración Original:")
print(oracion_original)
```

```
Oración Original:
drenched drenched drenched drenched drenched drenched drenched drenched drenc
```

Malísimo pero lo intentó.

Tengo la teoría que el error que me da, es que estoy tratando los outputs del modelo como si fueran de los embeddings que usé pero quizá los vectores que me da no son compatibles, quizá tendría que hacer uno personalizado o no sé.