Package definition

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount,
call drive.mount("/content/drive", force_remount=True).
```

```
import os
os.environ["KERAS_BACKEND"] = "tensorflow"

import pathlib, random, string, re
import numpy as np
import pandas as pd
import tensorflow as tf
import tensorflow.data as tf_data # type: ignore
import tensorflow.strings as tf_strings # type: ignore
import keras.ops as ops # type: ignore
from tensorflow import keras
from keras import layers
from keras.layers import TextVectorization # type: ignore
from tensorflow.keras.callbacks import CSVLogger # type: ignore
from tensorflow.keras.layers import TextVectorization # type: ignore
import matplotlib.pyplot as plt
import tensorflow.keras.backend as K # type: ignore
from tensorflow.keras.models import Model # type: ignore
print(tf.__version__)
```

```
2.18.0
```

Load pre-trained word embeddings

Se descargaron los 'embeddings' de GloVe (822M zip file).

1. Cargar los embeddings de GloVe

The archive contains text-encoded vectors of various sizes: 50-dimensional, 100-dimensional, 200-dimensional, 300-dimensional. We'll use the 100D ones.

Let's make a dict mapping words (strings) to their NumPy vector representation:

```
embedding_dim = 100

def load_glove_embeddings(path_glove, embedding_dim=embedding_dim):
    "Carga los embeddings de GloVe desde un archivo y los convierte en un
diccionario de palabras a vectores"
    embeddings_index = {}
    with open(path_glove, 'r', encoding='utf-8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs
    print(f"Se cargaron {len(embeddings_index)} palabras de GloVe.")
    return embeddings_index

# Cargar los embeddings GloVe de 50 dimensiones
#path_glove = "glove.6B/glove.6B.100d.txt"
path_glove = "/content/drive/MyDrive/ColabNotebooks/glove.6B.100d.txt"
embeddings_index = load_glove_embeddings(path_glove, 100)
```

```
Se cargaron 400001 palabras de GloVe.
```

2. Crear una matriz de embeddings para el vocabulario

Se requiere una matriz de embeddings que coincida con las palabras en el vocabulario, ya que solo las palabras que están en el vocabulario del modelo transformer deben tener un embedding correspondiente.

Carga de datos

```
text_file = "/content/drive/MyDrive/spa.txt"
#text_file = "spa-eng/spa.txt"
with open(text_file) as f:
    lines = f.read().split("\n")[:-1]
text_pairs = []
for line in lines:
    eng, spa = line.split("\t")
    spa = "[start] " + spa + " [end]"
    text_pairs.append((eng, spa))

random.shuffle(text_pairs)
num_val_samples = int(0.15 * len(text_pairs))
num_train_samples = len(text_pairs) - 2 * num_val_samples
train_pairs = text_pairs[:num_train_samples]
val_pairs = text_pairs[num_train_samples : num_train_samples +
num_val_samples]
test_pairs = text_pairs[num_train_samples + num_val_samples :]
```

```
# Verificar tamaños
print(f"Train: {len(train_pairs)}, Val: {len(val_pairs)}, Test:
{len(test_pairs)}")
```

```
Train: 83276, Val: 17844, Test: 17844
```

6.b Cambiar ngrams utilizaré 2, y 5 6.d Optimizadores: Adam y RSM

```
strip_chars = string.punctuation + "¿"
strip_chars = strip_chars.replace("[", "")
strip_chars = strip_chars.replace("]", "")

vocab_size = 15000
sequence_length = 20
batch_size = 64
# 6.b
ngrams = 5 #2 # 3, 5

def custom_standardization(input_string):
    lowercase = tf_strings.lower(input_string)
    return tf_strings.regex_replace(lowercase, "[%s]" %
re.escape(strip_chars), "")


eng_vectorization = TextVectorization(
    max_tokens=vocab_size,
    output_mode="int",
    output_sequence_length=sequence_length,
    ngrams=ngrams
)
spa_vectorization = TextVectorization(
    max_tokens=vocab_size,
    output_mode="int",
    output_sequence_length=sequence_length + 1,
    standardize=custom_standardization,
    ngrams=ngrams
)
train_eng_texts = [pair[0] for pair in train_pairs]
train_spa_texts = [pair[1] for pair in train_pairs]
eng_vectorization.adapt(train_eng_texts)
spa_vectorization.adapt(train_spa_texts)
```

2.1. Obtener el vocabulario y crear el índice de palabras:

Se tiene el vocabulario de español en *spa_vectorization*, el cual esta adaptado a partir del conjunto de datos de entrenamiento. Este se usara para crear la matriz de embeddings.

```
# Obtener el vocabulario de las palabras en español
spa_vocab = spa_vectorization.get_vocabulary()
# Crear un diccionario de índice de palabras (índice -> palabra)
spa_index_lookup = dict(zip(spa_vocab, range(len(spa_vocab))))

eng_vocab = eng_vectorization.get_vocabulary()
# Crear un diccionario de índice de palabras (índice -> palabra)
eng_index_lookup = dict(zip(eng_vocab, range(len(eng_vocab))))
```

Aquí spa_vocab es la lista de todas las palabras que spa_vectorization aprendió durante el adapt() del conjunto de datos en español. En el caso de traducción, usualmente se asigna un embedding a la secuencia de destino.

2. Crear la matriz de embeddings usando GloVe:

Se utilizará el diccionario de embeddings de GloVe cargado anteriormente y asignaremos sus vectores a las palabras de nuestro vocabulario de español.

Si deseas usar embeddings preentrenados de GloVe para ambas lenguas, deberías hacerlo tanto para el inglés como para el español, creando matrices de embeddings para ambos idiomas y luego usarlas en sus respectivas capas de embedding del modelo.

Embedding matrix

La función create_embedding_matrix crea una matriz de embeddings para las palabras en el vocabulario utilizando el diccionario de embeddings de GloVe cargado anteriormente. Para cada palabra en el vocabulario, la función intenta encontrar su vector correspondiente en el diccionario de embeddings de GloVe. Si la palabra se encuentra, su vector se asigna a la matriz de embeddings. Si no se encuentra la palabra, su vector se asigna a cero o al vector [UNK] para indicar que la palabra no está en el diccionario. Esta matriz se utilizará posteriormente en una capa de embeddings del modelo

```
# Prepare embedding matrix
#embedding_matrix = np.zeros((num_tokens, embedding_dim))
def create_embedding_matrix(vocab, embeddings_index, embedding_dim=100,
oov_token="[UNK]", pad_token="[PAD]"):
    """Crea una matriz de embeddings para las palabras en el vocabulario"""

    embedding_matrix = np.zeros((len(vocab), embedding_dim))  # +1 para el
token 0
    hits = 0
    misses = 0
    # Obtener el índice para [UNK] y [PAD]
    oov_token_idx = vocab.get(oov_token, None)
    pad_token_idx = vocab.get(pad_token, None)
```

```
    for word, i in vocab.items():
        if word == oov_token or word == pad_token:  # Si es el token [UNK]
o [PAD], lo dejamos en ceros
            continue  # No asignamos vectores a [UNK] y [PAD]
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
            hits += 1
        else:
            misses += 1
            # Si no se encuentra la palabra en GloVe, asigna el vector
[UNK] o lo deja en ceros
            if oov_token_idx is not None:
                embedding_matrix[oov_token_idx] = np.zeros(embedding_dim)
# Aquí asignamos un vector de ceros para [UNK]

    print(f"Converted {hits} words ({misses} misses)")
    return embedding_matrix

embedding_matrix_eng = create_embedding_matrix(eng_index_lookup,
embeddings_index, embedding_dim=100)
#embedding_matrix_spa = create_embedding_matrix(spa_index_lookup,
embeddings_index, embedding_dim=100)
```

```
Converted 3023 words (11976 misses)
```

```
from keras.layers import Embedding
num_tokens = len(eng_index_lookup)
embedding_dim = 100
embedding_layer = Embedding(
    num_tokens,
    embedding_dim,
    trainable=False,
)
embedding_layer.build((1,))
embedding_layer.set_weights([embedding_matrix_eng])
```

Load the transformer model

```
import keras.ops as ops

class TransformerEncoder(layers.Layer):
    def __init__(self, embed_dim, dense_dim, num_heads, **kwargs):
```

```
        super().__init__(**kwargs)
        self.embed_dim = embed_dim
        self.dense_dim = dense_dim
        self.num_heads = num_heads
        self.attention = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim
        )
        self.dense_proj = keras.Sequential(
            [
                layers.Dense(dense_dim, activation="relu"),
                layers.Dense(embed_dim),
            ]
        )
        self.layernorm_1 = layers.LayerNormalization()
        self.layernorm_2 = layers.LayerNormalization()
        self.supports_masking = True

    def call(self, inputs, mask=None):
        if mask is not None:
            padding_mask = ops.cast(mask[:, None, :], dtype="int32")
        else:
            padding_mask = None

        attention_output = self.attention(
            query=inputs, value=inputs, key=inputs,
attention_mask=padding_mask
        )
        proj_input = self.layernorm_1(inputs + attention_output)
        proj_output = self.dense_proj(proj_input)
        return self.layernorm_2(proj_input + proj_output)

    def get_config(self):
        config = super().get_config()
        config.update(
            {
                "embed_dim": self.embed_dim,
                "dense_dim": self.dense_dim,
                "num_heads": self.num_heads,
            }
        )
        return config


class PositionalEmbedding(layers.Layer):
    def __init__(self, sequence_length, vocab_size, embed_dim, **kwargs):
        super().__init__(**kwargs)
        self.token_embeddings = layers.Embedding(
            input_dim=vocab_size, output_dim=embed_dim
        )
        self.position_embeddings = layers.Embedding(
            input_dim=sequence_length, output_dim=embed_dim
        )
        self.sequence_length = sequence_length
        self.vocab_size = vocab_size
```

```
        self.embed_dim = embed_dim

    def call(self, inputs):
        length = ops.shape(inputs)[-1]
        positions = ops.arange(0, length, 1)
        embedded_tokens = self.token_embeddings(inputs)
        embedded_positions = self.position_embeddings(positions)
        return embedded_tokens + embedded_positions

    def compute_mask(self, inputs, mask=None):
        return ops.not_equal(inputs, 0)

    def get_config(self):
        config = super().get_config()
        config.update(
            {
                "sequence_length": self.sequence_length,
                "vocab_size": self.vocab_size,
                "embed_dim": self.embed_dim,
            }
        )
        return config


class TransformerDecoder(layers.Layer):
    def __init__(self, embed_dim, latent_dim, num_heads, **kwargs):
        super().__init__(**kwargs)
        self.embed_dim = embed_dim
        self.latent_dim = latent_dim
        self.num_heads = num_heads
        self.attention_1 = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim
        )
        self.attention_2 = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim
        )
        self.dense_proj = keras.Sequential(
            [
                layers.Dense(latent_dim, activation="relu"),
                layers.Dense(embed_dim),
            ]
        )
        self.layernorm_1 = layers.LayerNormalization()
        self.layernorm_2 = layers.LayerNormalization()
        self.layernorm_3 = layers.LayerNormalization()
        self.supports_masking = True

    def call(self, inputs, mask=None):
        inputs, encoder_outputs = inputs
        causal_mask = self.get_causal_attention_mask(inputs)

        if mask is None:
            inputs_padding_mask, encoder_outputs_padding_mask = None, None
        else:
```

```
            inputs_padding_mask, encoder_outputs_padding_mask = mask

        attention_output_1 = self.attention_1(
            query=inputs,
            value=inputs,
            key=inputs,
            attention_mask=causal_mask,
            query_mask=inputs_padding_mask,
        )
        out_1 = self.layernorm_1(inputs + attention_output_1)

        attention_output_2 = self.attention_2(
            query=out_1,
            value=encoder_outputs,
            key=encoder_outputs,
            query_mask=inputs_padding_mask,
            key_mask=encoder_outputs_padding_mask,
        )
        out_2 = self.layernorm_2(out_1 + attention_output_2)

        proj_output = self.dense_proj(out_2)
        return self.layernorm_3(out_2 + proj_output)

    def get_causal_attention_mask(self, inputs):
        input_shape = ops.shape(inputs)
        batch_size, sequence_length = input_shape[0], input_shape[1]
        i = ops.arange(sequence_length)[:, None]
        j = ops.arange(sequence_length)
        mask = ops.cast(i >= j, dtype="int32")
        mask = ops.reshape(mask, (1, input_shape[1], input_shape[1]))
        mult = ops.concatenate(
            [ops.expand_dims(batch_size, -1), ops.convert_to_tensor([1,
1])],
            axis=0,
        )
        return ops.tile(mask, mult)

    def get_config(self):
        config = super().get_config()
        config.update(
            {
                "embed_dim": self.embed_dim,
                "latent_dim": self.latent_dim,
                "num_heads": self.num_heads,
            }
        )
        return config
```

```
# Reconstruir el modelo
# Importa las clases personalizadas
```

```
#path_to_model = "English_to_Spanish_II.keras"
path_to_model =
"/content/drive/MyDrive/ColabNotebooks/English_to_Spanish_II.keras"
transformer = keras.models.load_model(path_to_model,
    custom_objects={
        "TransformerEncoder": TransformerEncoder,
        "PositionalEmbedding": PositionalEmbedding,
        "TransformerDecoder": TransformerDecoder,
    },
)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/layer.py:393:
UserWarning: `build()` was called on layer 'positional_embedding', however
the layer does not have a `build()` method implemented and it looks like it
has unbuilt state. This will cause the layer to be marked as built, despite
not being actually built, which may cause failures down the line. Make sure
to implement a proper `build()` method.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/keras/src/layers/layer.py:393:
UserWarning: `build()` was called on layer 'positional_embedding_1',
however the layer does not have a `build()` method implemented and it looks
like it has unbuilt state. This will cause the layer to be marked as built,
despite not being actually built, which may cause failures down the line.
Make sure to implement a proper `build()` method.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/keras/src/layers/layer.py:393:
UserWarning: `build()` was called on layer 'transformer_encoder', however
the layer does not have a `build()` method implemented and it looks like it
has unbuilt state. This will cause the layer to be marked as built, despite
not being actually built, which may cause failures down the line. Make sure
to implement a proper `build()` method.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/keras/src/layers/layer.py:393:
UserWarning: `build()` was called on layer 'transformer_decoder', however
the layer does not have a `build()` method implemented and it looks like it
has unbuilt state. This will cause the layer to be marked as built, despite
not being actually built, which may cause failures down the line. Make sure
to implement a proper `build()` method.
  warnings.warn(
```

```
# Obtener los pesos
weights = transformer.get_weights()  # Esto devuelve una lista con los
pesos de cada capa

# Ver la estructura de los pesos
# for i, w in enumerate(weights):
#     print(f"Peso {i}: Forma {w.shape}")
```

```
transformer.summary()
```

Model: "transformer"

Layer (type)	Output Shape	Param #	Connec
encoder_inputs (InputLayer)	(None, None)	0	-
decoder_inputs (InputLayer)	(None, None)	0	-
positional_embedding (PositionalEmbedding)	(None, None, 256)	3,845,120	encode
not_equal (NotEqual)	(None, None)	0	encode
positional_embedding_1 (PositionalEmbedding)	(None, None, 256)	3,845,120	decode
transformer_encoder (TransformerEncoder)	(None, None, 256)	3,155,456	positi not_eq
not_equal_1 (NotEqual)	(None, None)	0	decode
transformer_decoder (TransformerDecoder)	(None, None, 256)	5,259,520	positi transf not_eq not_eq
dropout_3 (Dropout)	(None, None, 256)	0	transf
dense_4 (Dense)	(None, None, 15000)	3,855,000	dropou

Total params: 39,920,434 (152.28 MB)

Trainable params: 19,960,216 (76.14 MB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 19,960,218 (76.14 MB)

Train the model

Entrenamiento con 50 epocas Se aumento el batch_size de 64 a 128

```
vocab_size = 1500
sequence_length = 20
batch_size = 128

def format_dataset(eng, spa):
    eng = eng_vectorization(eng)
    spa = spa_vectorization(spa)
    return (
        {
            "encoder_inputs": eng,
            "decoder_inputs": spa[:, :-1],
        },
        spa[:, 1:],
    )


def make_dataset(pairs):
    eng_texts, spa_texts = zip(*pairs)
    eng_texts = list(eng_texts)
    spa_texts = list(spa_texts)
    dataset = tf_data.Dataset.from_tensor_slices((eng_texts, spa_texts))
    dataset = dataset.batch(batch_size)
    dataset = dataset.map(format_dataset)
    return dataset.cache().shuffle(2048).prefetch(16)


train_ds = make_dataset(train_pairs)
val_ds = make_dataset(val_pairs)
```

```
import time
from tensorflow.keras.callbacks import CSVLogger, ModelCheckpoint,
EarlyStopping
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers import RMSprop

# Iniciar el tiempo
start_time = time.time()

# Callbacks para guardar logs, mejores modelos y evitar sobreentrenamiento
#csv_logger = CSVLogger("training_logs.csv", append=True)  # Guarda logs en
CSV
#csv_logger = CSVLogger("/content/drive/MyDrive/training_logs_1.csv",
append=True)  # Guarda logs en CSV
#csv_logger = CSVLogger("/content/drive/MyDrive/training_logs_2.csv",
append=True)  # Guarda logs en CSV
```

```
#csv_logger = CSVLogger("/content/drive/MyDrive/training_logs_3.csv",
append=True)  # Guarda logs en CSV
csv_logger = CSVLogger("/content/drive/MyDrive/training_logs_4.csv",
append=True)  # Guarda logs en CSV
checkpoint_callback = ModelCheckpoint(
    filepath="transformer_best.keras",  # Guarda el mejor modelo
    save_best_only=True,
    monitor="val_accuracy",
    mode="max"
)

early_stopping = EarlyStopping(
    monitor="val_loss",  # Detiene si la pérdida no mejora
    patience=8,  # Número de épocas sin mejora antes de detener
    restore_best_weights=True  # Restaura los mejores pesos encontrados
)

# 6.a Usar más de 30 épocas
new_epochs = 50

# 6.c Cambiar la tasa de aprendizaje
lr =1e-5  # Prueba con 1e-3, 5e-4, 1e-5, etc.
# 6.d Cambiar el optimizador
#optimizer = Adam(learning_rate=lr) # probar otros
optimizer = RMSprop(learning_rate=1e-4)

transformer.compile(
    optimizer=optimizer,
    loss=keras.losses.SparseCategoricalCrossentropy(ignore_class=0),
    metrics=["Accuracy"])


# Entrenamiento
history = transformer.fit(
    train_ds,
    epochs=new_epochs,
    validation_data=val_ds,
    callbacks=[csv_logger, checkpoint_callback, early_stopping]
)

# Guardar modelo completo
#transformer.save("transformer_updated.keras")

#transformer.save("/content/drive/MyDrive/transformer_ngram2_adam.keras")
#transformer.save("/content/drive/MyDrive/transformer_ngram5_adam.keras")
#transformer.save("/content/drive/MyDrive/transformer_ngram2_RMS.keras")
transformer.save("/content/drive/MyDrive/transformer_ngram5_RMS.keras")
# Guardar solo los pesos
#transformer.save_weights("transformer_weights.h5")

# Medir el tiempo total
elapsed_time = time.time() - start_time
print(f"Training completed in {elapsed_time:.2f} seconds")
```

```
Epoch 1/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m58s[0m
53ms/step - Accuracy: 0.3804 - loss: 5.8422 - val_Accuracy: 0.4922 -
val_loss: 3.5988
Epoch 2/50
[1m  1/651[0m [37m━━━━━━━━━━━━━━━━━━━━[0m [1m20s[0m 32ms/step -
Accuracy: 0.5012 - loss: 3.7690

/usr/local/lib/python3.11/dist-
packages/keras/src/callbacks/model_checkpoint.py:209: UserWarning: Can save
best model only with val_accuracy available, skipping.
  self._save_model(epoch=epoch, batch=None, logs=logs)


[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.4855 - loss: 3.7976 - val_Accuracy: 0.5379 -
val_loss: 2.9664
Epoch 3/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.5200 - loss: 3.2656 - val_Accuracy: 0.5693 -
val_loss: 2.5832
Epoch 4/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.5488 - loss: 2.8785 - val_Accuracy: 0.5892 -
val_loss: 2.3693
Epoch 5/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.5677 - loss: 2.6360 - val_Accuracy: 0.6024 -
val_loss: 2.2268
Epoch 6/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.5851 - loss: 2.4350 - val_Accuracy: 0.6158 -
val_loss: 2.0975
Epoch 7/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.5986 - loss: 2.2942 - val_Accuracy: 0.6264 -
val_loss: 2.0100
Epoch 8/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.6092 - loss: 2.1804 - val_Accuracy: 0.6324 -
val_loss: 1.9308
Epoch 9/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.6196 - loss: 2.0833 - val_Accuracy: 0.6379 -
val_loss: 1.8782
Epoch 10/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.6282 - loss: 2.0011 - val_Accuracy: 0.6485 -
val_loss: 1.7938
Epoch 11/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.6350 - loss: 1.9369 - val_Accuracy: 0.6561 -
```

```
val_loss: 1.7561
Epoch 12/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.6420 - loss: 1.8728 - val_Accuracy: 0.6582 -
val_loss: 1.7141
Epoch 13/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.6502 - loss: 1.8091 - val_Accuracy: 0.6704 -
val_loss: 1.6371
Epoch 14/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.6576 - loss: 1.7547 - val_Accuracy: 0.6724 -
val_loss: 1.6041
Epoch 15/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.6634 - loss: 1.7062 - val_Accuracy: 0.6758 -
val_loss: 1.5803
Epoch 16/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.6708 - loss: 1.6551 - val_Accuracy: 0.6860 -
val_loss: 1.5192
Epoch 17/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.6751 - loss: 1.6146 - val_Accuracy: 0.6867 -
val_loss: 1.5002
Epoch 18/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.6805 - loss: 1.5744 - val_Accuracy: 0.6937 -
val_loss: 1.4566
Epoch 19/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.6873 - loss: 1.5292 - val_Accuracy: 0.6956 -
val_loss: 1.4365
Epoch 20/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m16s[0m
24ms/step - Accuracy: 0.6914 - loss: 1.4947 - val_Accuracy: 0.7051 -
val_loss: 1.3974
Epoch 21/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.6966 - loss: 1.4622 - val_Accuracy: 0.7066 -
val_loss: 1.3740
Epoch 22/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.7012 - loss: 1.4259 - val_Accuracy: 0.7100 -
val_loss: 1.3541
Epoch 23/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.7049 - loss: 1.3984 - val_Accuracy: 0.7125 -
val_loss: 1.3271
Epoch 24/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.7094 - loss: 1.3684 - val_Accuracy: 0.7152 -
val_loss: 1.3114
Epoch 25/50
```

```
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.7143 - loss: 1.3355 - val_Accuracy: 0.7212 -
val_loss: 1.2857
Epoch 26/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.7188 - loss: 1.3082 - val_Accuracy: 0.7218 -
val_loss: 1.2652
Epoch 27/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7218 - loss: 1.2827 - val_Accuracy: 0.7200 -
val_loss: 1.2712
Epoch 28/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.7259 - loss: 1.2577 - val_Accuracy: 0.7259 -
val_loss: 1.2456
Epoch 29/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.7291 - loss: 1.2352 - val_Accuracy: 0.7281 -
val_loss: 1.2174
Epoch 30/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
24ms/step - Accuracy: 0.7328 - loss: 1.2112 - val_Accuracy: 0.7311 -
val_loss: 1.2107
Epoch 31/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7354 - loss: 1.1887 - val_Accuracy: 0.7345 -
val_loss: 1.1841
Epoch 32/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7392 - loss: 1.1628 - val_Accuracy: 0.7351 -
val_loss: 1.1787
Epoch 33/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7423 - loss: 1.1426 - val_Accuracy: 0.7380 -
val_loss: 1.1705
Epoch 34/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7459 - loss: 1.1250 - val_Accuracy: 0.7419 -
val_loss: 1.1445
Epoch 35/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7483 - loss: 1.1073 - val_Accuracy: 0.7364 -
val_loss: 1.1594
Epoch 36/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7505 - loss: 1.0880 - val_Accuracy: 0.7413 -
val_loss: 1.1468
Epoch 37/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7542 - loss: 1.0668 - val_Accuracy: 0.7422 -
val_loss: 1.1228
Epoch 38/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7568 - loss: 1.0518 - val_Accuracy: 0.7493 -
```

```
val_loss: 1.1008
Epoch 39/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7592 - loss: 1.0347 - val_Accuracy: 0.7482 -
val_loss: 1.0968
Epoch 40/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7622 - loss: 1.0171 - val_Accuracy: 0.7442 -
val_loss: 1.1087
Epoch 41/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7646 - loss: 1.0018 - val_Accuracy: 0.7529 -
val_loss: 1.0730
Epoch 42/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7665 - loss: 0.9893 - val_Accuracy: 0.7547 -
val_loss: 1.0639
Epoch 43/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7699 - loss: 0.9704 - val_Accuracy: 0.7539 -
val_loss: 1.0650
Epoch 44/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7718 - loss: 0.9573 - val_Accuracy: 0.7540 -
val_loss: 1.0578
Epoch 45/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7736 - loss: 0.9450 - val_Accuracy: 0.7566 -
val_loss: 1.0479
Epoch 46/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7756 - loss: 0.9335 - val_Accuracy: 0.7581 -
val_loss: 1.0425
Epoch 47/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7778 - loss: 0.9170 - val_Accuracy: 0.7575 -
val_loss: 1.0385
Epoch 48/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7794 - loss: 0.9042 - val_Accuracy: 0.7602 -
val_loss: 1.0330
Epoch 49/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7816 - loss: 0.8922 - val_Accuracy: 0.7594 -
val_loss: 1.0262
Epoch 50/50
[1m651/651[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m15s[0m
23ms/step - Accuracy: 0.7849 - loss: 0.8761 - val_Accuracy: 0.7628 -
val_loss: 1.0222
Training completed in 807.23 seconds
```

```
# Mostrar métricas finales
# print(history.history.keys())
final_acc = history.history["Accuracy"][-1]
final_val_acc = history.history["val_Accuracy"][-1]
final_loss = history.history["loss"][-1]
final_val_loss = history.history["val_loss"][-1]

print(f"Final Training Accuracy: {final_acc:.4f}")
print(f"Final Validation Accuracy: {final_val_acc:.4f}")
print(f"Final Training Loss: {final_loss:.4f}")
print(f"Final Validation Loss: {final_val_loss:.4f}")
```

```
Final Training Accuracy: 0.7826
Final Validation Accuracy: 0.7628
Final Training Loss: 0.8912
Final Validation Loss: 1.0222
```

Decoding test sentences

```
spa_vocab = spa_vectorization.get_vocabulary()
spa_index_lookup = dict(zip(range(len(spa_vocab)), spa_vocab))
max_decoded_sentence_length = 20


def decode_sequence(input_sentence):
    tokenized_input_sentence = eng_vectorization([input_sentence])
    decoded_sentence = "[start]"
    for i in range(max_decoded_sentence_length):
        tokenized_target_sentence = spa_vectorization([decoded_sentence])
[:, :-1]
        predictions = transformer(
            {
                "encoder_inputs": tokenized_input_sentence,
                "decoder_inputs": tokenized_target_sentence,
            }
        )

        # ops.argmax(predictions[0, i, :]) is not a concrete value for jax
here
        sampled_token_index = ops.convert_to_numpy(
            ops.argmax(predictions[0, i, :])
        ).item(0)
        sampled_token = spa_index_lookup[sampled_token_index]
        decoded_sentence += " " + sampled_token

        if sampled_token == "[end]":
            break
    return decoded_sentence
```

```
test_eng_texts = [pair[0] for pair in test_pairs]
for _ in range(10):
    input_sentence = random.choice(test_eng_texts)
    translated = decode_sequence(input_sentence)
    print(input_sentence, translated)
```

```
Go get changed. [start] ve [UNK] [end]
I hardly ever go to museums. [start] casi nunca voy a ir a [UNK] [end]
We missed our train. [start] nos [UNK] nuestro tren [end]
Honey, will you go shopping for me? [start] [UNK] que te [UNK] me [UNK]
[end]
I have something that you should see. [start] tengo algo que deberías ver
[end]
Tom likes sitting on the beach in the early morning. [start] a tom le gusta
en la playa en la playa [end]
Get out of my room! [start] [UNK] de mi habitación [end]
I'd like to work at the cafeteria. [start] me gustaría trabajar en el [UNK]
[end]
Spanish is spoken in Mexico. [start] el español se habla en méxico [end]
Will that briefcase hold many books? [start] [UNK] eso [UNK] muchos libros
[end]
```

```
# spa_vocab = spa_vectorization.get_vocabulary()
# spa_index_lookup = dict(zip(range(len(spa_vocab)), spa_vocab))
# max_decoded_sentence_length = 20


# def decode_sequence(input_sentence):
#     tokenized_input_sentence = eng_vectorization([input_sentence])
#     decoded_sentence = "[start]"
#     attention_weights = []  # Lista para almacenar los pesos de atención

#     for i in range(max_decoded_sentence_length):
#         tokenized_target_sentence = spa_vectorization([decoded_sentence])
[:, :-1]

#         # Assuming your transformer model only returns predictions
#         predictions = transformer(
#             {
#                 "encoder_inputs": tokenized_input_sentence,
#                 "decoder_inputs": tokenized_target_sentence,
#             }
#         )
#         # If you need attention weights, you might need to access them
#         # from a specific layer in your transformer model or modify its
output
#         # Example: attention_weights_1 =
```

```
transformer.get_layer('attention_layer_name').output
#         # Replace 'attention_layer_name' with the actual name of your
attention layer

#         # For now, we'll assign None to attention weights as they aren't
returned by the model
#         attention_weights_1 = None
#         attention_weights_2 = None

#         attention_weights.append((attention_weights_1,
attention_weights_2))  # Almacenar los pesos de atención

#         sampled_token_index = np.argmax(predictions[0, i, :])
#         sampled_token = spa_index_lookup[sampled_token_index]
#         decoded_sentence += " " + sampled_token

#         if sampled_token == "[end]":
#             break

#     return decoded_sentence, attention_weights
```

Metricas

- 6.e Cambiar las métricas.
- 6.f Se BLEU.
- 6.g Se utiliza Rouge

```
!pip install nltk
!pip install rouge-score
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-
packages (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-
packages (from nltk) (8.1.8)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-
packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-
packages (from nltk) (4.67.1)
Requirement already satisfied: rouge-score in
/usr/local/lib/python3.11/dist-packages (0.1.2)
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-
packages (from rouge-score) (1.4.0)
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-
```

```
packages (from rouge-score) (3.9.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-
packages (from rouge-score) (2.0.2)
Requirement already satisfied: six>=1.14.0 in
/usr/local/lib/python3.11/dist-packages (from rouge-score) (1.17.0)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-
packages (from nltk->rouge-score) (8.1.8)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-
packages (from nltk->rouge-score) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.11/dist-packages (from nltk->rouge-score)
(2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-
packages (from nltk->rouge-score) (4.67.1)
```

Rouge

```
from rouge_score import rouge_scorer

rouge_scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2'],
use_stemmer=True)
# Crear una lista de diccionarios para almacenar los resultados
rows = []
for test_pair in test_pairs[:100]:
    input_sentence = test_pair[0]
    reference_sentence = test_pair[1]
    translated_sentence = decode_sequence(input_sentence)
    translated_sentence = (
        translated_sentence.replace("[PAD]", "")
        .replace("[START]", "")
        .replace("[END]", "")
        .strip()
    )
    scores = rouge_scorer.score(reference_sentence, translated_sentence)
    # print(f"Input: {input_sentence}")
    # print(f"Reference: {reference_sentence}")
    # print(f"Translated: {translated_sentence}")
    # print(f"ROUGE-1 Precision: {scores['rouge1'].precision:.4f}, Recall:
{scores['rouge1'].recall:.4f}, F1-score: {scores['rouge1'].fmeasure:.4f}")
    # print(f"ROUGE-2 Precision: {scores['rouge2'].precision:.4f}, Recall:
{scores['rouge2'].recall:.4f}, F1-score: {scores['rouge2'].fmeasure:.4f}")
    # print("-" * 50)

    row = {
      "Input": input_sentence,
      "Reference": reference_sentence,
      "Translated": translated_sentence,
      "ROUGE-1 Precision": scores['rouge1'].precision,
      "ROUGE-1 Recall": scores['rouge1'].recall,
      "ROUGE-1 F1-score": scores['rouge1'].fmeasure,
      "ROUGE-2 Precision": scores['rouge2'].precision,
```

```
        "ROUGE-2 Recall": scores['rouge2'].recall,
        "ROUGE-2 F1-score": scores['rouge2'].fmeasure,
    }

rows.append(row)

# Create a Pandas DataFrame from the list of dictionaries
df = pd.DataFrame(rows)

# Export the DataFrame to a CSV file
#df.to_csv("rouge_scores_II.csv", index=False)
#df.to_csv("/content/drive/MyDrive/ColabNotebooks/rouge_scores_1.csv",
index=False)
#df.to_csv("/content/drive/MyDrive/ColabNotebooks/rouge_scores_2.csv",
index=False)
#df.to_csv("/content/drive/MyDrive/ColabNotebooks/rouge_scores_3.csv",
index=False)
df.to_csv("/content/drive/MyDrive/ColabNotebooks/rouge_scores_4.csv",
index=False)
```

BLEU

```
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction

# Crear una lista de diccionarios para almacenar los resultados
rows = []
smoother = SmoothingFunction().method1  # Suavizado para evitar BLEU=0 en
frases cortas

for test_pair in test_pairs[:100]:
    input_sentence = test_pair[0]
    reference_sentence = test_pair[1]
    translated_sentence = decode_sequence(input_sentence)
    translated_sentence = (
        translated_sentence.replace("[PAD]", "")
        .replace("[START]", "")
        .replace("[END]", "")
        .strip()
    )

    # Tokenizar las oraciones
    reference_tokens = [reference_sentence.split()]
    translated_tokens = translated_sentence.split()

    # Calcular BLEU con 1-gram, 2-gram, 3-gram y 4-gram
    bleu1 = sentence_bleu(reference_tokens, translated_tokens, weights=(1,
0, 0, 0), smoothing_function=smoother)
    bleu2 = sentence_bleu(reference_tokens, translated_tokens, weights=
(0.5, 0.5, 0, 0), smoothing_function=smoother)
    bleu3 = sentence_bleu(reference_tokens, translated_tokens, weights=
(0.33, 0.33, 0.33, 0), smoothing_function=smoother)
```

```
    bleu4 = sentence_bleu(reference_tokens, translated_tokens, weights=
(0.25, 0.25, 0.25, 0.25), smoothing_function=smoother)


    # Imprimir los resultados para cada par
    # print(f"Input: {input_sentence}")
    # print(f"Reference: {reference_sentence}")
    # print(f"Translated: {translated_sentence}")
    # print(f"BLEU-1: {bleu1:.4f}")
    # print(f"BLEU-2: {bleu2:.4f}")
    # print(f"BLEU-3: {bleu3:.4f}")
    # print(f"BLEU-4: {bleu4:.4f}")
    # print("-" * 50)


    row = {
        "Input": input_sentence,
        "Reference": reference_sentence,
        "Translated": translated_sentence,
        "BLEU-1": bleu1,
        "BLEU-2": bleu2,
        "BLEU-3": bleu3,
        "BLEU-4": bleu4,
    }
    rows.append(row)

# Crear DataFrame y exportar a CSV
#df = pd.DataFrame(rows)
#df.to_csv("/content/drive/MyDrive/ColabNotebooks/bleu_scores_1.csv",
index=False)
#df.to_csv("/content/drive/MyDrive/ColabNotebooks/bleu_scores_2.csv",
index=False)
#df.to_csv("/content/drive/MyDrive/ColabNotebooks/bleu_scores_3.csv",
index=False)
df.to_csv("/content/drive/MyDrive/ColabNotebooks/bleu_scores_4.csv",
index=False)
df.to_csv("bleu_scores.csv", index=False)
```

Layers

```
# Obtener las capas del modelo original
layer_outputs = [layer.output for layer in transformer.layers]  # Extraer
las salidas de cada capa

# Definir un nuevo modelo que devuelva las activaciones
activation_model = Model(inputs=transformer.input, outputs=layer_outputs)

# Verificar la estructura del modelo de activaciones
#activation_model.summary()
```

```
# Oración de prueba
ground_truth = "este es un libro interesante."
input_sentence = "this is an interesting book."
```

```
#input_sentence = "this is a beautiful tree"
# translated_sentence, attention_weights = decode_sequence(input_sentence)
translated_sentence = decode_sequence(input_sentence)
print(f"Translated: {translated_sentence}")
```

```
Translated: [start] este es un libro interesante [end]
```

```
def plot_attention_head(in_tokens, translated_tokens, attention, ax=None):
    """Visualiza la atención de una cabeza específica."""
    translated_tokens = translated_tokens[1:]  # Omite el primer token de
salida (start)

    if ax is None:
        fig, ax = plt.subplots(figsize=(10, 5))

    # Convertir a NumPy antes de graficar
    attention = np.array(attention)

    # Asegurar que la forma es válida
    if attention.ndim != 2:
        raise ValueError(f"Attention data must be 2D, but got shape
{attention.shape}")

    cax = ax.matshow(attention, cmap="viridis")

    # Etiquetas en los ejes
    ax.set_xticks(range(len(in_tokens)))
    ax.set_yticks(range(len(translated_tokens)))

    # Convertir los tokens a texto si son tensores
    if isinstance(in_tokens, tf.Tensor):
        in_tokens = in_tokens.numpy()
    if isinstance(translated_tokens, tf.Tensor):
        translated_tokens = translated_tokens.numpy()

    ax.set_xticklabels([label.decode("utf-8") for label in
in_tokens.numpy()], rotation=90)
    ax.set_yticklabels([label.decode("utf-8") for label in
```

```
translated_tokens.numpy()])

    plt.colorbar(cax)
    ax.set_title("Mapa de Atención (Cabeza específica)")
```

```
def plot_attention_weights(sentence, translated_tokens, attention_weights):
    """Visualiza todas las cabezas de atención."""
    in_tokens = eng_vectorization([sentence]).numpy()  # Entrada del
codificador (español)

    fig = plt.figure(figsize=(16, 8))

    # Verificar que `attention_weights` tiene datos
    if not isinstance(attention_weights, list) or len(attention_weights) ==
0:
        print("No attention weights available.")
        return

    for h, head in enumerate(attention_weights):
        if head is None or np.shape(head) == ():
            print(f"Attention head {h} is empty or incorrect.")
            continue  # Saltar esta cabeza si está vacía

        ax = fig.add_subplot(2, 4, h + 1)
        plot_attention_head(in_tokens, translated_tokens, np.array(head),
ax)
        ax.set_xlabel(f'Head {h + 1}')

    plt.tight_layout()
    plt.show()
```

```
# Traducir y obtener los pesos de atención
# translated_sentence, attention_weights = decode_sequence(input_sentence)
```

```
# print(f"Attention weights type: {type(attention_weights)}")
# print(f"Attention weights shape: {np.shape(attention_weights)}")
# print(f"First attention head shape: {np.shape(attention_weights[0]) if
len(attention_weights) > 0 else 'Empty'}")
```

```
# Visualizar los pesos de atención
# plot_attention_weights(input_sentence, translated_sentence,
```

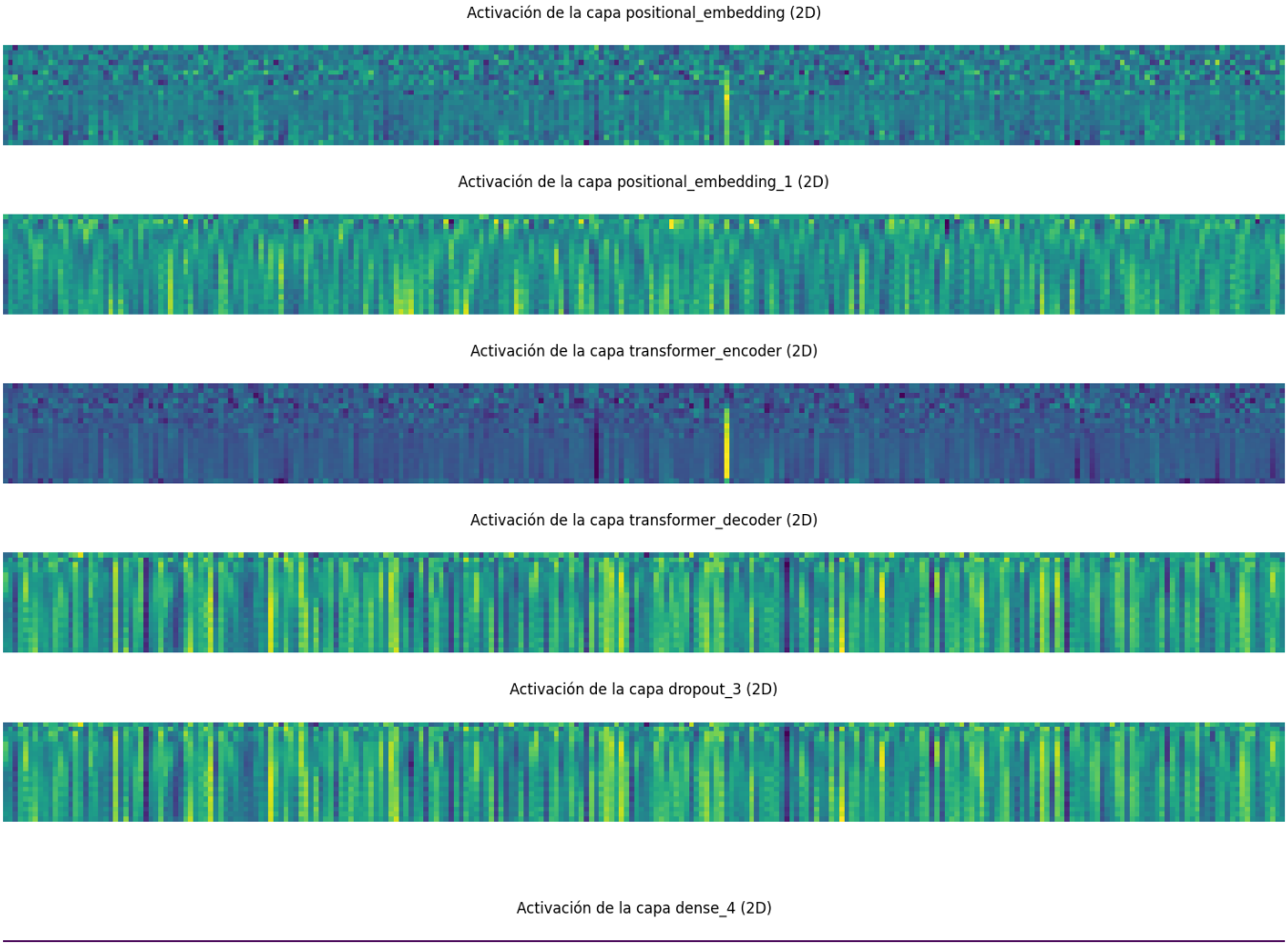
```
attention_weights[2])
```

Intento sin attention_weights

```
input_sentence = "this is a beautiful tree"
tokenized_encoder_input = eng_vectorization([input_sentence])

# Inicializar la entrada del decoder con el token "[start]"
start_token = spa_vectorization(["[start]"])[:, :-1]  # Quita el último
token

activations = activation_model.predict({
    "encoder_inputs": tokenized_encoder_input,
    "decoder_inputs": start_token
})

# Información de las activaciones
for i, activation in enumerate(activations):
    print(f"🔹 Capa {i + 1}: {transformer.layers[i].name}")
    print(f"   Forma de la activación: {activation.shape}\n")
```

```
□[1m1/1□[0m □[32m━━━━━━━━━━━━━━━━━━━━□[0m□[37m□[0m □[1m2s□[0m 2s/step
🔹 Capa 1: encoder_inputs
   Forma de la activación: (1, 20)

🔹 Capa 2: decoder_inputs
   Forma de la activación: (1, 20)

🔹 Capa 3: positional_embedding
   Forma de la activación: (1, 20, 256)

🔹 Capa 4: positional_embedding_1
   Forma de la activación: (1, 20, 256)

🔹 Capa 5: transformer_encoder
   Forma de la activación: (1, 20, 256)

🔹 Capa 6: transformer_decoder
   Forma de la activación: (1, 20, 256)

🔹 Capa 7: dropout_3
   Forma de la activación: (1, 20, 256)

🔹 Capa 8: dense_4
   Forma de la activación: (1, 20, 15000)
```

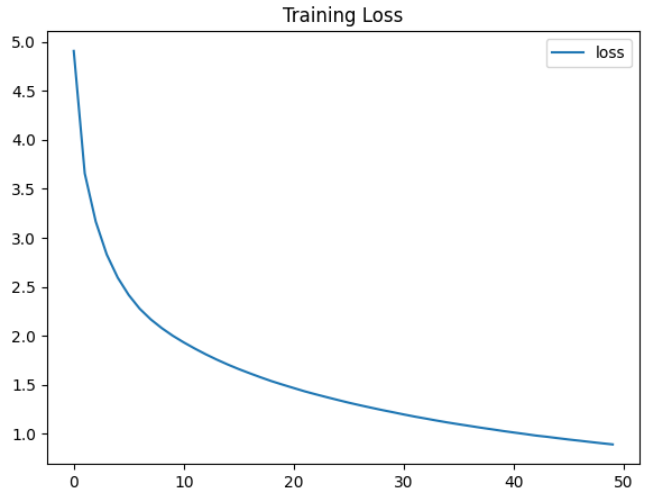
```
import numpy as np
import matplotlib.pyplot as plt

# Elegir las capas a visualizar
layer_indices = [2, 3, 4, 5, 6, 7]

# Crear una figura con una subgráfica por capa
fig, axes = plt.subplots(len(layer_indices), 1, figsize=(15, 2 *
len(layer_indices)))

palabras_interes = ["love", "moon", "range"]  # Palabras de interés

for idx, layer_idx in enumerate(layer_indices):
    activation_selected = activations[layer_idx]
    ax = axes[idx]

    if activation_selected.ndim == 1:
        ax.plot(activation_selected[0])
        ax.set_title(f"Activación de la capa
{transformer.layers[layer_idx].name} (1D)")
    else:
        ax.matshow(activation_selected[0], cmap="viridis")
        ax.set_title(f"Activación de la capa
{transformer.layers[layer_idx].name} (2D)")

        # Obtener tokens y palabras correspondientes
        if layer_idx == 2:  # Capa del encoder
            tokens = eng_vectorization.get_vocabulary()
            indices = tokenized_encoder_input[0].numpy()
            words = [tokens[i] for i in indices if i < len(tokens)]
        elif layer_idx == 3:  # Capa del decoder
            tokens = spa_vectorization.get_vocabulary()
            indices = start_token[0].numpy()
            words = [tokens[i] for i in indices if i < len(tokens)]
        else:
            words = None

        if words:
            indices_palabras_interes = [i for i, word in enumerate(words)
if word in palabras_interes]
            if indices_palabras_interes:
                ax.set_xticks(np.arange(len(words)))
                ax.set_xticklabels(words, rotation=45, ha="right")  #
Mostrar palabras en el eje x

    ax.axis("off")

plt.tight_layout()
plt.show()
```

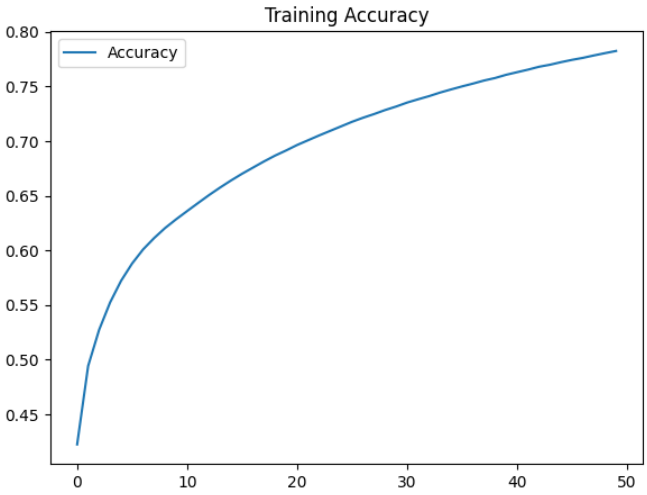


Resultados

```
df_results = pd.read_csv("/content/drive/MyDrive/training_logs_4.csv",
usecols=["Accuracy", "loss"]).head(100)
losses = df_results["loss"].values
accuracies = df_results["Accuracy"].values
```

```
import matplotlib.pyplot as plt
%matplotlib inline
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,5))
# plot some data
ax1.plot(losses, label='loss')
#plt.plot(results.history['val_loss'], label='val_loss')
ax1.set_title('Training Loss')
ax1.legend()
# accuracies
ax2.plot(accuracies, label='Accuracy')
#plt.plot(results.history['val_accuracy_fn'], label='val_acc')
ax2.set_title('Training Accuracy')
ax2.legend()
plt.show()
```


Training Loss
loss
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
10
20
30
40
50


Training Accuracy
Accuracy
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0
10
20
30
40
50