English-to-Spanish translation with a sequence-to-sequence Transformer

Se realizó el preprocesamiento de datos para entrenar un modelo de traducción automática de inglés a español. Primero, se cargó un archivo de texto que contenía pares de oraciones en ambos idiomas y se prepararon los datos agregando etiquetas de inicio y fin a las frases en español.

Luego, se dividieron los pares en conjuntos de entrenamiento, validación y prueba. Se llevó a cabo una limpieza de caracteres no deseados y se configuraron capas de vectorización para transformar las frases en secuencias numéricas, adaptándolas al vocabulario del conjunto de entrenamiento. Finalmente, se organizaron los datos en conjuntos tf.data.Dataset.

```
!pip install tensorflow==2.18
```

```
Collecting tensorflow==2.18
  Downloading tensorflow-2.18.0-cp311-cp311-manylinux_2_17_x86_64.manylinux201
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/loc
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.1
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.1
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/pyth
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.1
Collecting tensorboard<2.19,>=2.18 (from tensorflow==2.18)
  Downloading tensorboard-2.18.0-py3-none-any.whl.metadata (1.6 kB)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /usr/local/lib/python3.
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-
Collecting ml-dtypes<0.5.0,>=0.4.0 (from tensorflow==2.18)
  Downloading ml_dtypes-0.4.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_
```

```
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/l
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.1
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyt
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/d
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/d
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/pyth
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-p
Downloading tensorflow-2.18.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 615.4/615.4 MB 2.7 MB/s eta 0:00:
Downloading ml_dtypes-0.4.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 62.5 MB/s eta 0:00:00
Downloading tensorboard-2.18.0-py3-none-any.whl (5.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.5/5.5 MB 85.2 MB/s eta 0:00:00
Installing collected packages: ml-dtypes, tensorboard, tensorflow
  Attempting uninstall: ml-dtypes
    Found existing installation: ml_dtypes 0.5.1
    Uninstalling ml_dtypes-0.5.1:
      Successfully uninstalled ml_dtypes-0.5.1
  Attempting uninstall: tensorboard
    Found existing installation: tensorboard 2.19.0
    Uninstalling tensorboard-2.19.0:
      Successfully uninstalled tensorboard-2.19.0
Successfully installed ml-dtypes-0.4.1 tensorboard-2.18.0 tensorflow-2.18.0
```

```python
import os
import pathlib
import random
import string
import re
import numpy as np
import tensorflow as tf
from keras.layers import TextVectorization

file_path = "/content/spa.txt"
with open(file_path) as f:
    lines = f.read().split("\n")[:-1]

# Preparar los pares de traducción
```

```
text_pairs = []
for line in lines:
    eng, spa = line.split("\t")
    spa = "[start] " + spa + " [end]"
    text_pairs.append((eng, spa))

# Mostrar ejemplos aleatorios de los pares
for _ in range(5):
    print(random.choice(text_pairs))

# Dividir los pares en entrenamiento, validación y prueba
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 :]

print(f"{len(text_pairs)} total pairs")
print(f"{len(train_pairs)} training pairs")
print(f"{len(val_pairs)} validation pairs")
print(f"{len(test_pairs)} test pairs")

# Definir caracteres a eliminar
strip_chars = string.punctuation + "¿"
strip_chars = strip_chars.replace("[", "")
strip_chars = strip_chars.replace("]", "")

# Configuración del modelo
vocab_size = 15000
sequence_length = 22
batch_size = 64

# Estandarización personalizada
def custom_standardization(input_string):
    lowercase = tf.strings.lower(input_string)
    return tf.strings.regex_replace(lowercase, "[%s]" % re.escape(strip_chars), "

# Crear la capa de vectorización
eng_vectorization = TextVectorization(
    max_tokens=vocab_size,
    output_mode="int",
    output_sequence_length=sequence_length,
)
spa_vectorization = TextVectorization(
```

```
    max_tokens=vocab_size,
    output_mode="int",
    output_sequence_length=sequence_length + 1,
    standardize=custom_standardization,
)

# Adaptar el vectorizador
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)

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)

# Crear los datasets de entrenamiento y validación
train_ds = make_dataset(train_pairs)
val_ds = make_dataset(val_pairs)

# Mostrar las formas de los datos de ejemplo
for inputs, targets in train_ds.take(1):
    print(f'inputs["encoder_inputs"].shape: {inputs["encoder_inputs"].shape}')
    print(f'inputs["decoder_inputs"].shape: {inputs["decoder_inputs"].shape}')
```

```
('I can put things in a box.', '[start] Puedo poner las cosas en una caja. [er
('I want to know what happened to your car.', '[start] Quiero saber qué le pas
('He is painting a picture.', '[start] Él está pintando un cuadro. [end]')
('It is fact that he wants to visit Egypt.', '[start] Es un hecho de que quier
('Tom moaned.', '[start] Tom gimió. [end]')
118964 total pairs
83276 training pairs
17844 validation pairs
17844 test pairs
inputs["encoder_inputs"].shape: (64, 22)
inputs["decoder_inputs"].shape: (64, 22)
```

Construyendo el modelo

Se implementó un modelo basado en la arquitectura Transformer para la traducción automática. Tal que, se definieron tres componentes principales: el codificador, el decodificador y la capa de incrustaciones posicionales. El codificador se encargó de procesar las entradas utilizando mecanismos de atención multi-cabezal y proyecciones densas, normalizando las salidas en cada etapa. La capa de incrustaciones posicionales permitió representar la información de posición dentro de las secuencias de entrada.

Por otro lado, el decodificador incorporó atención causal para modelar dependencias secuenciales y mecanismos de atención cruzada para procesar las salidas del codificador.

```
import keras
from tensorflow.keras import layers

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
```

Aquí, se construyó la arquitectura completa del modelo Transformer definiendo sus entradas, conexiones y salidas. Primero, se establecieron las dimensiones de incrustación y los parámetros del modelo, como la cantidad de cabezales de atención y el tamaño de las capas densas. Luego, se implementó el codificador, el cual recibió las entradas de texto y las transformó mediante incrustaciones posicionales y la capa de TransformerEncoder, generando representaciones intermedias del texto de entrada.

Consecuentemente, se definió el decodificador, que tomó como entrada la secuencia objetivo y las representaciones del codificador, aplicando capas de incrustaciones y TransformerDecoder para generar predicciones.

Finalmente, se integraron ambos módulos en un modelo Transformer completo, estableciendo las relaciones entre el codificador, el decodificador y utilizando una capa densa con activación softmax para obtener las probabilidades de las palabras de salida.

```
embed_dim = 256
latent_dim = 2048
num_heads = 8

encoder_inputs = keras.Input(shape=(None,), dtype="int64", name="encoder_inputs")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(encoder_inputs)
encoder_outputs = TransformerEncoder(embed_dim, latent_dim, num_heads)(x)
encoder = keras.Model(encoder_inputs, encoder_outputs)

decoder_inputs = keras.Input(shape=(None,), dtype="int64", name="decoder_inputs")
encoded_seq_inputs = keras.Input(shape=(None, embed_dim), name="decoder_state_inp
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(decoder_inputs)
x = TransformerDecoder(embed_dim, latent_dim, num_heads)([x, encoder_outputs])
x = layers.Dropout(0.5)(x)
decoder_outputs = layers.Dense(vocab_size, activation="softmax")(x)
decoder = keras.Model([decoder_inputs, encoded_seq_inputs], decoder_outputs)

transformer = keras.Model(
    {"encoder_inputs": encoder_inputs, "decoder_inputs": decoder_inputs},
    decoder_outputs,
    name="transformer",
)
```

```
!apt-get install libcudnn8
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  libcudnn8
0 upgraded, 1 newly installed, 0 to remove and 29 not upgraded.
Need to get 444 MB of archives.
After this operation, 1,099 MB of additional disk space will be used.
Get:1 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_
Fetched 444 MB in 5s (90.7 MB/s)
Selecting previously unselected package libcudnn8.
(Reading database ... 126209 files and directories currently installed.)
Preparing to unpack .../libcudnn8_8.9.7.29-1+cuda12.2_amd64.deb ...
Unpacking libcudnn8 (8.9.7.29-1+cuda12.2) ...
Setting up libcudnn8 (8.9.7.29-1+cuda12.2) ...
```

```
!ls /content/glove
```

```
cooccur.c  demo.sh  eval  glove.c  LICENSE  makefile  README  shuffle.c  vocal
```

Entrenando el modelo

Aquí, se generó un resumen de la arquitectura del modelo con transformer.summary(). Luego, se compiló el modelo utilizando el optimizador RMSprop y la función de pérdida SparseCategoricalCrossentropy, ignorando la clase 0 (evitando que afectara el cálculo del error).

Así mismo, se definió la métrica de precisión para evaluar el desempeño durante el entrenamiento. Por último, al modelo se le entrenó durante 50 épocas con los conjuntos de datos de entrenamiento y validación (originalmente de 1, luego de 100 y finalmente de 50).

```
epochs = 50

transformer.summary()
transformer.compile(
    "rmsprop",
    loss=keras.losses.SparseCategoricalCrossentropy(ignore_class=0),
    metrics=["accuracy"],
)
transformer.fit(train_ds, epochs=epochs, validation_data=val_ds)
```

Model: "transformer"

Layer (type)	Output Shape	Param #	Conne
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	posit: not_e

(TransformerEncoder)			not_e
not_equal_1 (NotEqual)	(None, None)	0	decod
transformer_decoder (TransformerDecoder)	(None, None, 256)	5,259,520	posit trans not_e not_e
dropout_3 (Dropout)	(None, None, 256)	0	trans
dense_4 (Dense)	(None, None, 15000)	3,855,000	dropo

Total params: 19,960,216 (76.14 MB)
Trainable params: 19,960,216 (76.14 MB)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 96s 73ms/step - accuracy: 0.0921 - loss: 5.1651
Epoch 2/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 111s 56ms/step - accuracy: 0.1703 - loss: 2.928
Epoch 3/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 85s 59ms/step - accuracy: 0.1894 - loss: 2.4242
Epoch 4/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 81s 59ms/step - accuracy: 0.1990 - loss: 2.2027
Epoch 5/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 60s 59ms/step - accuracy: 0.2062 - loss: 2.0584
Epoch 6/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 58s 57ms/step - accuracy: 0.2098 - loss: 1.9780
Epoch 7/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 57ms/step - accuracy: 0.2129 - loss: 1.9317
Epoch 8/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 58s 57ms/step - accuracy: 0.2166 - loss: 1.8700
Epoch 9/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 82s 57ms/step - accuracy: 0.2179 - loss: 1.8409
Epoch 10/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 82s 57ms/step - accuracy: 0.2204 - loss: 1.8027
Epoch 11/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 57ms/step - accuracy: 0.2217 - loss: 1.7947
Epoch 12/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 57ms/step - accuracy: 0.2230 - loss: 1.7684
Epoch 13/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 57ms/step - accuracy: 0.2245 - loss: 1.7612
Epoch 14/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 56ms/step - accuracy: 0.2257 - loss: 1.7263
Epoch 15/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 58s 57ms/step - accuracy: 0.2274 - loss: 1.7079
Epoch 16/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 58s 57ms/step - accuracy: 0.2280 - loss: 1.6938
Epoch 17/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 82s 57ms/step - accuracy: 0.2283 - loss: 1.6966
Epoch 18/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 56ms/step - accuracy: 0.2300 - loss: 1.6650
```

```
Epoch 19/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 83s 57ms/step - accuracy: 0.2313 - loss: 1.6508
Epoch 20/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 58s 57ms/step - accuracy: 0.2313 - loss: 1.6383
Epoch 21/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 82s 57ms/step - accuracy: 0.2333 - loss: 1.6165
Epoch 22/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 58s 57ms/step - accuracy: 0.2330 - loss: 1.6174
Epoch 23/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 58s 57ms/step - accuracy: 0.2344 - loss: 1.5963
Epoch 24/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 58s 57ms/step - accuracy: 0.2345 - loss: 1.5913
Epoch 25/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 58s 57ms/step - accuracy: 0.2356 - loss: 1.5806
Epoch 26/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 58s 57ms/step - accuracy: 0.2363 - loss: 1.5709
Epoch 27/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 60s 59ms/step - accuracy: 0.2364 - loss: 1.5728
Epoch 28/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 80s 57ms/step - accuracy: 0.2379 - loss: 1.5448
Epoch 29/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 81s 57ms/step - accuracy: 0.2378 - loss: 1.5410
Epoch 30/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 82s 57ms/step - accuracy: 0.2389 - loss: 1.5265
Epoch 31/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 84s 59ms/step - accuracy: 0.2394 - loss: 1.5141
Epoch 32/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 82s 59ms/step - accuracy: 0.2395 - loss: 1.5247
Epoch 33/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 80s 57ms/step - accuracy: 0.2400 - loss: 1.5142
Epoch 34/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 56ms/step - accuracy: 0.2411 - loss: 1.4937
Epoch 35/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 57ms/step - accuracy: 0.2414 - loss: 1.4887
Epoch 36/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 56ms/step - accuracy: 0.2413 - loss: 1.4933
Epoch 37/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 57ms/step - accuracy: 0.2419 - loss: 1.4746
Epoch 38/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 60s 59ms/step - accuracy: 0.2428 - loss: 1.4721
Epoch 39/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 57ms/step - accuracy: 0.2427 - loss: 1.4694
Epoch 40/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 82s 57ms/step - accuracy: 0.2435 - loss: 1.4560
Epoch 41/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 57ms/step - accuracy: 0.2435 - loss: 1.4606
Epoch 42/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 57ms/step - accuracy: 0.2440 - loss: 1.4436
Epoch 43/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 82s 57ms/step - accuracy: 0.2442 - loss: 1.4437
```

```
Epoch 44/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 56ms/step - accuracy: 0.2454 - loss: 1.4303
Epoch 45/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 57ms/step - accuracy: 0.2448 - loss: 1.4301
Epoch 46/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 56ms/step - accuracy: 0.2453 - loss: 1.4255
Epoch 47/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 57ms/step - accuracy: 0.2458 - loss: 1.4101
Epoch 48/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 57ms/step - accuracy: 0.2453 - loss: 1.4207
Epoch 49/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 57s 57ms/step - accuracy: 0.2456 - loss: 1.4177
Epoch 50/50
1013/1013 ━━━━━━━━━━━━━━━━━━━━ 60s 59ms/step - accuracy: 0.2462 - loss: 1.4064
<keras.src.callbacks.history.History at 0x7ef17021ef90>
```

Guardando el modelo

```
transformer.save("transformer_model.keras")
```

Decodificando oraciones de prueba

Aquí se implementó una función de depuración; en cada ejecución, se seleccionó una oración de prueba en inglés y se tokenizó utilizando eng_vectorization(). Luego, se inició la generación de la oración traducida con la palabra clave [start], iterando 20 veces para predecir la siguiente palabra en cada paso.

Durante cada iteración, se tokenizó la oración generada hasta el momento con spa_vectorization(); se verificó que la tokenización no produjera una secuencia vacía o mal formada, lo que permitió detectar errores antes de alimentar los datos al modelo. Luego, se pasaron las entradas al Transformer para obtener predicciones sobre la siguiente palabra en la secuencia.

Para determinar el siguiente token, se tomó el índice con la mayor probabilidad en la salida del modelo y se convirtió en su palabra correspondiente utilizando spa_index_lookup. Si el modelo generaba el token [end], el proceso terminaba.

El proceso se repitió para tres oraciones seleccionadas aleatoriamente de test_eng_texts, lo que ayudó a evaluar el rendimiento y la estabilidad del modelo en diferentes ejemplos.

```
def debug_decode_sequence(input_sentence):
    print(f"\nProcesando: {input_sentence}")

    # Tokenización entrada
    tokenized_input_sentence = eng_vectorization([input_sentence])
    print(f"Tokenizado (entrada): {tokenized_input_sentence}")

    decoded_sentence = "[start]"
    max_steps = 20
    for i in range(max_steps):

        tokenized_target_sentence = spa_vectorization([decoded_sentence])

        # Verificación de que tokenized_target_sentence no está vacío o mal forma
        if tokenized_target_sentence is None or tokenized_target_sentence.shape[1
            print("¡Error! La secuencia de salida está vacía o no se generó corre
            break

        print(f"Iteración {i} - Tokenizado (objetivo): {tokenized_target_sentence
        print(f"Forma de tokenized_target_sentence: {tokenized_target_sentence.sh

        predictions = transformer(
            {
```

```
                "encoder_inputs": tokenized_input_sentence,
                "decoder_inputs": tokenized_target_sentence,
            }
        )

        print(f"Predicciones en la iteración {i}: {predictions}")

        # Tomamos el índice con la mayor probabilidad
        sampled_token_index = ops.convert_to_numpy(
            ops.argmax(predictions[0, i, :])
        ).item()

        print(f"Índice de token generado: {sampled_token_index}") # Imprimir el í
        sampled_token = spa_index_lookup.get(sampled_token_index, "[unk]")  # Bus
        print(f"Token generado: {sampled_token}")


        decoded_sentence += " " + sampled_token # Actualizamos la secuencia decod

        # Si encontramos el token de fin, terminamos
        if sampled_token == "[end]":
            print("Se encontró [end]. Terminando.")
            break

    print(f"Resultado final: {decoded_sentence}")
    return decoded_sentence

for i in range(3):
    input_sentence = random.choice(test_eng_texts)
    debug_decode_sequence(input_sentence)
```

```
Procesando: What is your name?
Tokenizado (entrada): [[ 24   8  28 231   0   0   0   0   0   0   0   0   0
    0   0]]
Iteración 0 - Tokenizado (objetivo): [[2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (
Forma de tokenized_target_sentence: (1, 21)
Predicciones en la iteración 0: [[[7.2720664e-05 7.5809876e-05 7.3737407e-05 .
   5.2781023e-05 4.9948478e-05]
  [6.1155079e-05 5.0418363e-05 6.5946158e-05 ... 6.1235463e-05
   6.2531588e-05 6.9855734e-05]
  [6.1060935e-05 5.2562045e-05 6.5753789e-05 ... 6.8091897e-05
   6.0969982e-05 5.2276730e-05]
  ...
  [7.6135431e-05 4.2834130e-05 6.5628483e-05 ... 5.5641744e-05
   5.9386934e-05 6.2920546e-05]
```

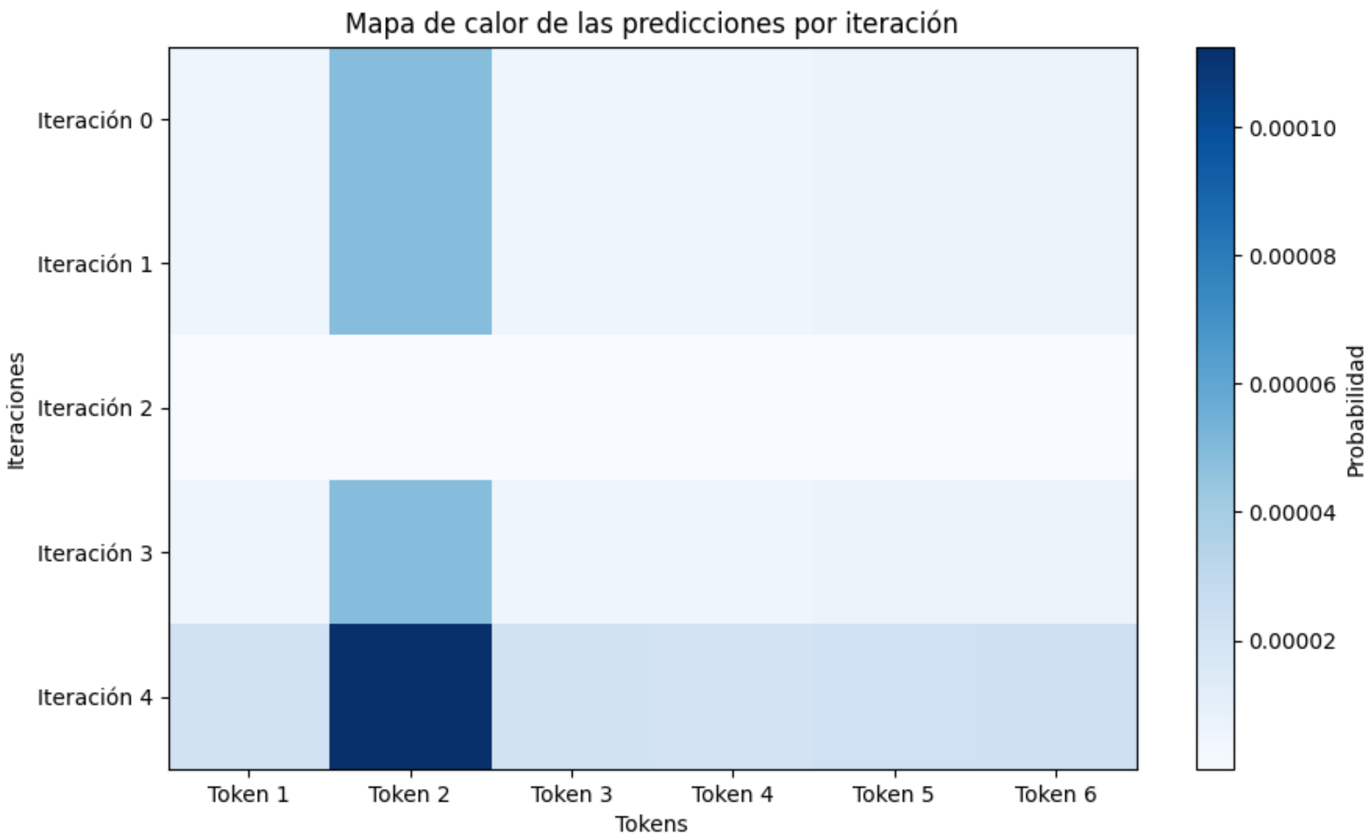
```
  [8.2506849e-05 4.3079992e-05 6.3054897e-05 ... 5.9578415e-05
   6.2636886e-05 6.3876920e-05]
  [6.1540042e-05 4.6741898e-05 6.2301675e-05 ... 6.3475163e-05
   6.4071246e-05 6.4517990e-05]]]
Índice de token generado: 984
Token generado: fin
Iteración 1 - Tokenizado (objetivo): [[  2 390   0   0   0   0   0   0   0   (
    0   0   0]]
Forma de tokenized_target_sentence: (1, 21)
Predicciones en la iteración 1: [[[7.2720664e-05 7.5809876e-05 7.3737407e-05 .
   5.2781023e-05 4.9948478e-05]
  [8.5253516e-05 5.8876394e-05 8.9178931e-05 ... 4.8529226e-05
   6.7076347e-05 7.0252252e-05]
  [6.1085993e-05 5.2571857e-05 6.5724322e-05 ... 6.8156784e-05
   6.0981576e-05 5.2273921e-05]
  ...
  [7.6166369e-05 4.2844989e-05 6.5618355e-05 ... 5.5706769e-05
   5.9391648e-05 6.2892606e-05]
  [8.2558676e-05 4.3096967e-05 6.3019921e-05 ... 5.9643960e-05
   6.2636689e-05 6.3867352e-05]
  [6.1595754e-05 4.6761099e-05 6.2266663e-05 ... 6.3547661e-05
   6.4062733e-05 6.4486543e-05]]]
Índice de token generado: 7465
Token generado: [unk]
Iteración 2 - Tokenizado (objetivo): [[  2 390   1   0   0   0   0   0   0   (
    0   0   0]]
Forma de tokenized_target_sentence: (1, 21)
Predicciones en la iteración 2: [[[7.2720664e-05 7.5809876e-05 7.3737407e-05 .
   5.2781023e-05 4.9948478e-05]
  [8.5253516e-05 5.8876394e-05 8.9178931e-05 ... 4.8529226e-05
   6.7076347e-05 7.0252252e-05]
  [7.0259113e-05 7.2445808e-05 6.5504537e-05 ... 7.5633332e-05
   6.2662970e-05 4.6057488e-05]
  ...
  [7.6134165e-05 4.2839631e-05 6.5636741e-05 ... 5.5748813e-05
   5.9406582e-05 6.2904423e-05]
  [8.2522689e-05 4.3095326e-05 6.3030115e-05 ... 5.9685142e-05
   6.2623352e-05 6.3906227e-05]
  [6.1568084e-05 4.6741825e-05 6.2295236e-05 ... 6.3603416e-05
   6.4046420e-05 6.4522857e-05]]]
Índice de token generado: 2122
Token generado: [unk]
Iteración 3 - Tokenizado (objetivo): [[  2 390   1   1   0   0   0   0   0   (
    0   0   0]]
```

Por último, se generó un mapa de calor para visualizar cómo evolucionan las probabilidades de predicción de los tokens a lo largo de varias iteraciones.

```
# Tomamos algunos
```

```
predicciones_iteraciones = [
    [5.2318273e-06, 4.8568301e-05, 5.2295122e-06, 4.8669726e-06, 5.7674274e-06, 5.7
    [5.2317832e-06, 4.8568636e-05, 5.2294886e-06, 4.8670058e-06, 5.7674447e-06, 5.7
    [2.8557724e-11, 7.8660047e-12, 2.8548303e-11, 2.4043476e-11, 3.0711513e-11, 3.0
    [5.2317851e-06, 4.8568465e-05, 5.2294308e-06, 4.8670076e-06, 5.7674247e-06, 5.7
    [2.1522730e-05, 1.1249992e-04, 2.1515711e-05, 2.0532158e-05, 2.2606364e-05, 2.3
]

predicciones_array = np.array(predicciones_iteraciones)
plt.figure(figsize=(10, 6))
plt.imshow(predicciones_array, cmap="Blues", aspect="auto", interpolation="nearest"
plt.colorbar(label="Probabilidad")
plt.xticks(np.arange(predicciones_array.shape[1]), ["Token 1", "Token 2", "Token 3"
plt.yticks(np.arange(predicciones_array.shape[0]), [f"Iteración {i}" for i in range
plt.title('Mapa de calor de las predicciones por iteración')
plt.xlabel('Tokens')
plt.ylabel('Iteraciones')
plt.show()
```


Mapa de calor de las predicciones por iteración
Iteraciones
Iteración 0
Iteración 1
Iteración 2
Iteración 3
Iteración 4
Token 1
Token 2
Token 3
Token 4
Token 5
Token 6
Tokens
Probabilidad
0.00010
0.00008
0.00006
0.00004
0.00002