Adaptación de *English-to-Spanish NMT by fchollet*

El siguiente modelo se entreno desde cero. Posteriormente se hará el tratamiento con los embeddings' de GloVe.

- Se implementa el modelo transformador de secuencia a secuencia
- Entrenamiento para realizar traducciones.
- Se almacena el modelo preentrenado para el uso futuro.

```
# We set the backend to TensorFlow. The code works with
# both `tensorflow` and `torch`. It does not work with JAX
# due to the behavior of `jax.numpy.tile` in a jit scope
# (used in `TransformerDecoder.get_causal_attention_mask()`:
# `tile` in JAX does not support a dynamic `reps` argument.
# You can make the code work in JAX by wrapping the
# inside of the `get_causal_attention_mask` method in
# a decorator to prevent jit compilation:
# `with jax.ensure_compile_time_eval():`.
import os

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

import pathlib
import random
import string
import re
import numpy as np
import tensorflow as tf
import tensorflow.data as tf_data
import tensorflow.strings as tf_strings

import keras
from keras import layers
from keras import layers
from keras import ops
from keras.layers import TextVectorization
```

Descargar el conjunto de datos de Anki *English-to-Spanish*

```
#=text_file = keras.utils.get_file(
    # fname="spa-eng.zip",
    #
origin="http://storage.googleapis.com/download.tensorflow.org/data/spa-
eng.zip",
    # extract=True,
# )
```

```
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).
```

```
#text_file = pathlib.Path(text_file).parent / "spa-eng" / "spa.txt"
text_file = "/content/drive/MyDrive/spa.txt"
#text_file = "spa-eng/spa.txt"
```

Analisis de los datos.

Cada línea del conjunto de datos contiene una oración en ingles y su correspondiente oración en español. La oración en inglés es la *source sequence*(oración fuente) y la oración en español es la *target sequence* (oración destino). Para cada token se antepone "[start]" al inicio y se agrega "[end]" al final de la oración en español.

```
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))
```

Vistazo a los pares de oraciones inglés-español.

```
for _ in range(5):
    print(random.choice(text_pairs))
```

```
('The cold air revived Tom.', '[start] El aire helado resucitó a Tom.
[end]')
("She's the most beautiful woman in the world.", '[start] Ella es la mujer
más hermosa del mundo. [end]')
("I'd like to marry a girl who likes to play video games.", '[start] Quiero
casarme con una mujer a la que le gusten los videojuegos. [end]')
('Tom has a degree in biology.', '[start] Tom tiene un grado en biología.
[end]')
('Tom buttoned his shirt.', '[start] Tomás se abotonó la camisa. [end]')
```

A continuación se dividen los pares de oraciones en datos de prueba y de entrenamiento de forma aleatoria.

```
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")
```

```
118964 total pairs
83276 training pairs
17844 validation pairs
17844 test pairs
```

Vectorización

Se utilizaron dos instancias de la capa *TextVectorization* para vectorizar los datos de texto una para inglés y otra para español, para convertir las cadenas originales en secuencias de números enteros donde cada entero representa el índice de una palabra en un vocabulario.

La capa de inglés utilizará la estandarización de cadenas predeterminada (eliminando los caracteres de puntuación) y el esquema de división (dividir por espacios), mientras que la capa de español utilizará una estandarización personalizada para remover carácteres de puntuación propios del lenguaje español.

Nota: En un modelo de traducción automática de producción, no recomendaría eliminar los caracteres de puntuación en ninguno de los dos idiomas. En su lugar, recomendaría convertir cada carácter de puntuación en su propio token, lo cual se podría lograr proporcionando una función de división personalizada a la capa TextVectorization.

```
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 = 3
```

```
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)
```

A continuación, se da formato a los conjuntos de datos.

En cada paso de entrenamiento, el modelo intentará predecir las palabras objetivo N+1 (y posteriores) utilizando la oración fuente y las palabras objetivo de 0 a N.

Para ello, el conjunto de datos de entrenamiento generará una tupla (entradas, objetivos), donde:

- Entradas es un diccionario con las claves encoder_inputs y decoder_inputs.
- encoder_inputs es la oración fuente vectorizada.
- decoder_inputs es la oración objetivo con las palabras de 0 a N utilizadas para predecir la palabra N+1 (y posteriores) en la oración objetivo.
- Objetivo es la oración objetivo desplazada por un paso: proporciona las siguientes palabras en la oración objetivo, lo que el modelo intentará predecir.

```
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)
```

Formas de la secuencia: Se tienen lotes de 64 pares y todas las secuencias tienen una longitud de 20 pasos

```
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}')
    print(f"targets.shape: {targets.shape}")
```

```
inputs["encoder_inputs"].shape: (64, 20)
inputs["decoder_inputs"].shape: (64, 20)
targets.shape: (64, 20)
```

Construcción del modelo

Este modelo Transformer secuencia a secuencia consta de un *TransformerEncoder* y un *TransformerDecoder* encadenados. Para que el modelo reconozca el orden de las palabras, se utiliza una capa de *PositionalEmbedding*.

La secuencia fuente envía al TransformerEncoder el cual genera una nueva representación de la misma. Esta nueva representación pasa al TransformerDecoder, junto con la secuencia objetivo hasta el momento (palabras objetivo de 0 a N). El TransformerDecoder intenta predecir las siguientes palabras en la secuencia objetivo (N+1 y posteriores).

Un detalle clave que lo hace posible es el enmascaramiento causal (véase el método get_causal_attention_mask() en el TransformerDecoder). El TransformerDecoder ve todas las secuencias a la vez, por lo que debemos asegurarnos de que solo utilice la información de los tokens objetivo de 0 a N al predecir el token N+1 (de lo contrario, podría utilizar información del futuro, lo que daría como resultado un modelo que no se pueda utilizar en el momento de hacer la inferencia.

```
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
```

A continuación se ensambla el modelo de extremo a extremo.

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

# Entrada y salida del encoder y decoder
encoder_inputs = layers.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 = layers.Input(shape=(None,), dtype="int64",
name="decoder_inputs")
encoded_seq_inputs = layers.Input(shape=(None, embed_dim),
name="decoder_state_inputs")
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)

# Modelo Transformer final
transformer = keras.Model(
    {"encoder_inputs": encoder_inputs, "decoder_inputs": decoder_inputs},
    decoder_outputs,
    name="transformer",
)

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

```
# import tensorflow as tf
# print(tf.__version__)

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: 19,960,216 (76.14 MB)
```

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

```
Non-trainable params: 0 (0.00 B)
```

Entrenando el modelo

Se utiliza la precisión como una forma rápida de monitorear el progreso del entrenamiento con los datos de validación. También para la traducción automática suelen utilizar otras métricas como *BLEU*, *ROUGE*.

Para que el modelo converja realmente se debe entrenar durante al menos 30 épocas.

```
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

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=5,  # 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 = 100

# 6.c Cambiar la tasa de aprendizaje
lr = 1e-4  # 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("Englis_to_Spanish_II.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")
```

```
# 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}")
```

Decodificación de oraciones de prueba

A continuación se demostrará cómo traducir oraciones nuevas en inglés. Simplemente introducimos en el modelo la oración vectorizada en inglés y el token de destino "[start]". Luego, generamos repetidamente el siguiente token hasta llegar al token "[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]"
    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)
```

```
Children learn to respond to rhythmical sounds from a very young age.
[start] los niños [UNK] a [UNK] de [UNK] muy [UNK] [end]
The water has been cut off. [start] el [UNK] ha estado [UNK] [end]
Did you remember to close the windows? [start] te [UNK] a [UNK] la mesa
[end]
We received word of his death. [start] [UNK] la noche de su madre [end]
You're not my friend anymore. [start] no eres mi amigo [end]
You would look stupid wearing your mother's dress. [start] te [UNK] [UNK]
tu [UNK] de la [UNK] [end]
I feel much better already. [start] me siento mucho mejor [end]
How long will you have to wait? [start] cómo te [UNK] que tienes que [UNK]
[end]
We're closed. [start] estamos [UNK] [end]
Make good use of your time. [start] [UNK] bien tu tiempo [end]
```

Metricas

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

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

```
Collecting rouge-score
  Downloading rouge_score-0.1.2.tar.gz (17 kB)
  Preparing metadata (setup.py) ... ▯[?25l▯[?25hdone
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)
Building wheels for collected packages: rouge-score
  Building wheel for rouge-score (setup.py) ... [?25l[?25hdone
  Created wheel for rouge-score: filename=rouge_score-0.1.2-py3-none-
any.whl size=24935
sha256=112aec7735bd5e427a965e3051ae4290c476e3805ae708d735f7e74e4e6c3b35
  Stored in directory:
/root/.cache/pip/wheels/1e/19/43/8a442dc83660ca25e163e1bd1f89919284ab0d0c14
75475148
Successfully built rouge-score
Installing collected packages: rouge-score
Successfully installed rouge-score-0.1.2
```

```python
from rouge_score import rouge_scorer
import pandas as pd
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[:10]:
    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_f.csv",
index=False)
```

```
Input: It's not safe to drive without wearing a seatbelt.
Reference: [start] No es seguro conducir sin usar un cinturón de seguridad.
[end]
Translated: [start] no es muy bien de [UNK] sin un [UNK] [end]
ROUGE-1 Precision: 0.6364, Recall: 0.5385, F1-score: 0.5833
ROUGE-2 Precision: 0.2000, Recall: 0.1667, F1-score: 0.1818
--------------------------------------------------
Input: Your pants are unzipped.
Reference: [start] Tienes la cremallera de los pantalones bajada. [end]
Translated: [start] tu [UNK] son [UNK] [end]
ROUGE-1 Precision: 0.3333, Recall: 0.2222, F1-score: 0.2667
ROUGE-2 Precision: 0.0000, Recall: 0.0000, F1-score: 0.0000
--------------------------------------------------
Input: Do you want to wait?
Reference: [start] ¿Querés esperar? [end]
Translated: [start] quieres [UNK] [end]
ROUGE-1 Precision: 0.5000, Recall: 0.4000, F1-score: 0.4444
ROUGE-2 Precision: 0.0000, Recall: 0.0000, F1-score: 0.0000
--------------------------------------------------
Input: That's how he likes it.
Reference: [start] Así es como le gusta. [end]
Translated: [start] eso es como le gusta [end]
ROUGE-1 Precision: 0.8571, Recall: 0.8571, F1-score: 0.8571
ROUGE-2 Precision: 0.6667, Recall: 0.6667, F1-score: 0.6667
--------------------------------------------------
Input: You should turn off the light before going to sleep.
Reference: [start] Deberías apagar la luz antes de irte a dormir. [end]
Translated: [start] deberías [UNK] la [UNK] antes de [UNK] [end]
ROUGE-1 Precision: 0.7000, Recall: 0.5833, F1-score: 0.6364
ROUGE-2 Precision: 0.3333, Recall: 0.2727, F1-score: 0.3000
--------------------------------------------------
Input: His dog follows him wherever he goes.
```

```
Reference: [start] Su perro le sigue adondequiera que vaya. [end]
Translated: [start] su perro [UNK] que él se [UNK] [end]
ROUGE-1 Precision: 0.5556, Recall: 0.5556, F1-score: 0.5556
ROUGE-2 Precision: 0.2500, Recall: 0.2500, F1-score: 0.2500
--------------------------------------------------
Input: Tom has three uncles.
Reference: [start] Tom tiene tres tíos. [end]
Translated: [start] tom tiene tres años [end]
ROUGE-1 Precision: 0.8571, Recall: 0.8571, F1-score: 0.8571
ROUGE-2 Precision: 0.6667, Recall: 0.6667, F1-score: 0.6667
--------------------------------------------------
Input: You're witty.
Reference: [start] Eres ingenioso. [end]
Translated: [start] eres [UNK] [end]
ROUGE-1 Precision: 0.7500, Recall: 0.7500, F1-score: 0.7500
ROUGE-2 Precision: 0.3333, Recall: 0.3333, F1-score: 0.3333
--------------------------------------------------
Input: Our team lost the first game.
Reference: [start] Nuestro equipo perdió el primer encuentro. [end]
Translated: [start] nuestro tren se ha estado el tren [end]
ROUGE-1 Precision: 0.4444, Recall: 0.5000, F1-score: 0.4706
ROUGE-2 Precision: 0.1250, Recall: 0.1429, F1-score: 0.1333
--------------------------------------------------
Input: Do you know where he lives?
Reference: [start] ¿Sabéis dónde vive? [end]
Translated: [start] sabes dónde vive [end]
ROUGE-1 Precision: 0.8333, Recall: 0.7143, F1-score: 0.7692
ROUGE-2 Precision: 0.6000, Recall: 0.5000, F1-score: 0.5455
--------------------------------------------------
```

```python
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[:10]:
    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_f.csv",
index=False)
df.to_csv("bleu_scores.csv", index=False)
```

```
Input: It's not safe to drive without wearing a seatbelt.
Reference: [start] No es seguro conducir sin usar un cinturón de seguridad.
[end]
Translated: [start] no es muy bien de [UNK] sin un [UNK] [end]
BLEU-1: 0.4981
BLEU-2: 0.0674
BLEU-3: 0.0370
BLEU-4: 0.0269
--------------------------------------------------
```

```
Input: Your pants are unzipped.
Reference: [start] Tienes la cremallera de los pantalones bajada. [end]
Translated: [start] tu [UNK] son [UNK] [end]
BLEU-1: 0.2022
BLEU-2: 0.0495
BLEU-3: 0.0344
BLEU-4: 0.0294
--------------------------------------------------
Input: Do you want to wait?
Reference: [start] ¿Querés esperar? [end]
Translated: [start] quieres [UNK] [end]
BLEU-1: 0.5000
BLEU-2: 0.1291
BLEU-3: 0.0964
BLEU-4: 0.0955
--------------------------------------------------
Input: That's how he likes it.
Reference: [start] Así es como le gusta. [end]
Translated: [start] eso es como le gusta [end]
BLEU-1: 0.7143
BLEU-2: 0.4880
BLEU-3: 0.3662
BLEU-4: 0.1858
--------------------------------------------------
Input: You should turn off the light before going to sleep.
Reference: [start] Deberías apagar la luz antes de irte a dormir. [end]
Translated: [start] deberías [UNK] la [UNK] antes de [UNK] [end]
BLEU-1: 0.4449
BLEU-2: 0.2110
BLEU-3: 0.0817
BLEU-4: 0.0511
--------------------------------------------------
Input: His dog follows him wherever he goes.
Reference: [start] Su perro le sigue adondequiera que vaya. [end]
Translated: [start] su perro [UNK] que él se [UNK] [end]
BLEU-1: 0.4444
BLEU-2: 0.0745
BLEU-3: 0.0443
BLEU-4: 0.0339
--------------------------------------------------
Input: Tom has three uncles.
Reference: [start] Tom tiene tres tíos. [end]
Translated: [start] tom tiene tres años [end]
BLEU-1: 0.6667
BLEU-2: 0.3651
BLEU-3: 0.1522
BLEU-4: 0.1027
--------------------------------------------------
Input: You're witty.
Reference: [start] Eres ingenioso. [end]
Translated: [start] eres [UNK] [end]
BLEU-1: 0.5000
BLEU-2: 0.1291
BLEU-3: 0.0964
```

```
BLEU-4: 0.0955
--------------------------------------------------
Input: Our team lost the first game.
Reference: [start] Nuestro equipo perdió el primer encuentro. [end]
Translated: [start] nuestro tren se ha estado el tren [end]
BLEU-1: 0.3333
BLEU-2: 0.0645
BLEU-3: 0.0403
BLEU-4: 0.0316
--------------------------------------------------
Input: Do you know where he lives?
Reference: [start] ¿Sabéis dónde vive? [end]
Translated: [start] sabes dónde vive [end]
BLEU-1: 0.6000
BLEU-2: 0.1225
BLEU-3: 0.0814
BLEU-4: 0.0707
--------------------------------------------------
```

```
```