Transformer Challenge

Iniciamos cargando todas las paqueterias necesarias para poder correr el codigo, toda esta primera parte es casi un copia y pega del codigo que se menciono en el reto, con taan solo pequenas modificaciones para que funcionara en estee entorno

In [1]:
```
import os

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

import pathlib
import random
import string
import re
import numpy as np

import tensorflow.data as tf_data
import tensorflow.strings as tf_strings
import tensorflow as tf
import keras
from keras import layers
from keras import ops
from keras.layers import TextVectorization
```

```
2025-03-22 21:55:05.604865: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1742702105.622199   26083 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
E0000 00:00:1742702105.627607   26083 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
2025-03-22 21:55:05.645814: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

Descargamos los datos y les hacemos moddificaciones para su uso

In [2]:
```
text_file = keras.utils.get_file(
    fname="spa-eng.zip",
    origin="http://storage.googleapis.com/download.tensorflow.org/data/spa-eng.zip",
    extract=True,
)
text_file = pathlib.Path(text_file).parent / "spa-eng" / "spa.txt"
```

In [3]:
```
text_file
```

Out[3]:
```
PosixPath('/home/galo/.keras/datasets/spa-eng/spa.txt')
```

In [4]:
```
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))
```

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

```
('He was absent from the meeting.', '[start] Él no estaba en la reunión. [end]')
('Tom ran like crazy to catch up with Mary.', '[start] Tom corrió como loco para alcanzar a Mary. [end]')
("Tom doesn't know anything about Mary.", '[start] Tom no sabe nada acerca de Mary. [end]')
("We're in a hurry.", '[start] Estamos apurados. [end]')
('My aunt has been dead for two years.', '[start] Mi tía lleva muerta dos años. [end]')
```

Se dividen los distintos conjuntos de entrenamiento, validacion y prueba

In [6]:
```
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
```

Por lo que entinedo en esta parte se crean y modifican lso ngramas, en este caso es un ngrama de 20

In [7]:
```
strip_chars = string.punctuation + "¿"
strip_chars = strip_chars.replace("[", "")
strip_chars = strip_chars.replace("]", "")

vocab_size = 15000
sequence_length = 20
batch_size = 64


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

```
)
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)
```

```
I0000 00:00:1742702110.032666   26083 gpu_device.cc:2022] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 2246 MB memory:  -> device: 0, name: NVIDIA GeForce GTX 1650, pci bus id: 0000:01:00.0, compute capability: 7.5
```

In [8]:
```
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)
```

In [9]:
```
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)
2025-03-22 21:55:12.188955: I tensorflow/core/framework/local_rendezvous.cc:405] Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

Se crean todos los metodos necesarios, como los encoders, decoders y se establece el transformer, se agrego la etiquyeta, para poder guardar el modelo de forma sencilla, guardando los layers tambien, @keras.saving.register_keras_serializable(package="MyLayers")

In [10]:
```
import keras.ops as ops

@keras.saving.register_keras_serializable(package="MyLayers")
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

@keras.saving.register_keras_serializable(package="MyLayers")
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

@keras.saving.register_keras_serializable(package="MyLayers")
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
```

In [11]:

```
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_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)

transformer = keras.Model(
```

```
    {"encoder_inputs": encoder_inputs, "decoder_inputs": decoder_inputs},
    decoder_outputs,
    name="transformer",
)
```

Por practiciddad de momento, se usara una solaa epoca, ya que por epoca en mi computadora se tarda alrededor de 2 minutos.

In [12]:
```
epochs = 1  # This should be at least 30 for convergence

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 #	Connected to
encoder_inputs (InputLayer)	(None, None)	0	-
decoder_inputs (InputLayer)	(None, None)	0	-
positional_embeddi… (PositionalEmbeddi…	(None, None, 256)	3,845,120	encoder_inputs[0…
not_equal (NotEqual)	(None, None)	0	encoder_inputs[0…
positional_embeddi… (PositionalEmbeddi…	(None, None, 256)	3,845,120	decoder_inputs[0…
transformer_encoder (TransformerEncode…	(None, None, 256)	3,155,456	positional_embed… not_equal[0][0]
not_equal_1 (NotEqual)	(None, None)	0	decoder_inputs[0…
transformer_decoder (TransformerDecode…	(None, None, 256)	5,259,520	positional_embed… transformer_enco… not_equal_1[0][0… not_equal[0][0]
dropout_3 (Dropout)	(None, None, 256)	0	transformer_deco…
dense_4 (Dense)	(None, None, 15000)	3,855,000	dropout_3[0][0]

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

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

Non-trainable params: 0 (0.00 B)

```
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1742702122.165922   26162 service.cc:148] XLA service 0x7fc48c002520 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
I0000 00:00:1742702122.165987   26162 service.cc:156]   StreamExecutor device (0): NVIDIA GeForce GTX 1650, Compute Capability 7.5
2025-03-22 21:55:22.371359: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:268] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
W0000 00:00:1742702122.507637   26162 assert_op.cc:38] Ignoring Assert operator compile_loss/sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/assert_equal_1/Assert/As
sert
I0000 00:00:1742702123.338537   26162 cuda_dnn.cc:529] Loaded cuDNN version 90800
2025-03-22 21:55:34.018984: I external/local_xla/xla/stream_executor/cuda/cuda_asm_compiler.cc:397] ptxas warning : Registers are spilled to local memory in function 'input_add_reduce_f
usion_4', 24 bytes spill stores, 24 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'input_add_reduce_fusion_2', 24 bytes spill stores, 24 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'input_add_reduce_fusion_1', 24 bytes spill stores, 24 bytes spill loads

I0000 00:00:1742702134.102374   26162 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
 187/1302 ━━━━━━━━━━━━━━━━━━━━ 1:37 88ms/step - accuracy: 0.0577 - loss: 6.6291
W0000 00:00:1742702150.990698   26160 assert_op.cc:38] Ignoring Assert operator compile_loss/sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/assert_equal_1/Assert/As
sert
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.1039 - loss: 5.0724
W0000 00:00:1742702273.892857   26164 assert_op.cc:38] Ignoring Assert operator compile_loss/sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/assert_equal_1/Assert/As
sert
W0000 00:00:1742702277.738210   26164 assert_op.cc:38] Ignoring Assert operator compile_loss/sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/assert_equal_1/Assert/As
sert
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 169s 114ms/step - accuracy: 0.1039 - loss: 5.0717 - val_accuracy: 0.1947 - val_loss: 2.9103
```

Out[12]: <keras.src.callbacks.history.History at 0x7fc4e47a33d0>

In [13]:
```
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(30):
    input_sentence = random.choice(test_eng_texts)
    translated = decode_sequence(input_sentence)
```

Guardamos el modelo ya con los pesos y layyers customs

In [14]:
```
transformer.save('transformer_2_1.keras')
```

Cargamos el modelo anteriormente entrenado y lo usamos para predecir diferentes oraciones

In [15]:
```
nuevo_modelo = tf.keras.models.load_model('transformer_2_1.keras')
```

```
/home/galo/Diplomado/python/TF/lib/python3.10/site-packages/keras/src/layers/layer.py:395: 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 d
own the line. Make sure to implement a proper `build()` method.
  warnings.warn(
/home/galo/Diplomado/python/TF/lib/python3.10/site-packages/keras/src/layers/layer.py:395: UserWarning: `build()` was called on layer 'positional_embedding_1', however the layer does no
t 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(
/home/galo/Diplomado/python/TF/lib/python3.10/site-packages/keras/src/layers/layer.py:395: UserWarning: `build()` was called on layer 'transformer_encoder', however the layer does not h
ave 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 do
wn the line. Make sure to implement a proper `build()` method.
  warnings.warn(
/home/galo/Diplomado/python/TF/lib/python3.10/site-packages/keras/src/layers/layer.py:395: UserWarning: `build()` was called on layer 'transformer_decoder', however the layer does not h
ave 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 do
wn the line. Make sure to implement a proper `build()` method.
  warnings.warn(
```

In [16]:
```
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 = nuevo_modelo(
            {
                "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(30):
    input_sentence = random.choice(test_eng_texts)
    translated = decode_sequence(input_sentence)
```

Este modelo por ser solo una epoca es extremadamente malo al predecir las traducciones

In [17]:
```
print(input_sentence)
print(translated)
```

```
He slept all day.
[start] Él se puso todo el día [end]
```

Modelo preentrenado

Ahora empezamos a cargar los word embeddings preentrenados

In [18]:
```
path_to_glove_file = "glove.6B/glove.6B.100d.txt"

embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print("Found %s word vectors." % len(embeddings_index))
```

```
Found 400000 word vectors.
```

In [19]:
```
voc = eng_vectorization.get_vocabulary()
word_index = dict(zip(voc, range(len(voc))))
```

In [20]:
```
num_tokens = 12077
embedding_dim = 100
hits = 0
misses = 0
embed_dim = 100

embedding_matrix = np.zeros((num_tokens, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:

        embedding_matrix[i] = embedding_vector
        hits += 1
    else:
        misses += 1
print("Converted %d words (%d misses)" % (hits, misses))
```

```
Converted 11691 words (318 misses)
```

In [21]:
```
vocab_size = 12077
```

In [22]:
```
from tensorflow.keras.layers import Embedding
encoder_inputs = keras.Input(shape=(None,), dtype="int64", name="encoder_inputs")
x =  Embedding(input_dim=vocab_size, output_dim=embedding_dim,
                            weights=[embedding_matrix], trainable=False)(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_inputs")
x =  Embedding(input_dim=vocab_size, output_dim=embedding_dim,
                            weights=[embedding_matrix], trainable=False)(decoder_inputs)
x = TransformerDecoder(embed_dim, latent_dim, num_heads)([x, encoder_outputs])
decoder_outputs = layers.Dense(vocab_size, activation="softmax")(x)

decoder = keras.Model([decoder_inputs, encoded_seq_inputs], decoder_outputs)

transformer1 = keras.Model(
    {"encoder_inputs": encoder_inputs, "decoder_inputs": decoder_inputs},
    decoder_outputs,
    name="transformer",
)
```

Ya con los pesos de los embeddings y con tan solo una epoca se logra un accuracy del 60% demostrando lo util que son estos weights ya entrenados

In [23]:
```
epochs = 1  # This should be at least 30 for convergence

transformer1.summary()
transformer1.compile(
    "rmsprop",
    loss=keras.losses.SparseCategoricalCrossentropy(ignore_class=0),
    metrics=["accuracy"],
)
transformer1.fit(train_ds, epochs=epochs, validation_data=val_ds)
```

Model: "transformer"

Layer (type)	Output Shape	Param #	Connected to
encoder_inputs (InputLayer)	(None, None)	0	-
decoder_inputs (InputLayer)	(None, None)	0	-
embedding_8 (Embedding)	(None, None, 100)	1,207,700	encoder_inputs[0…
embedding_9 (Embedding)	(None, None, 100)	1,207,700	decoder_inputs[0…
transformer_encode… (TransformerEncode…	(None, None, 100)	734,648	embedding_8[0][0]
transformer_decode… (TransformerDecode…	(None, None, 100)	1,057,348	embedding_9[0][0… transformer_enco…
dense_13 (Dense)	(None, None, 12077)	1,219,777	transformer_deco…

Total params: 5,427,173 (20.70 MB)

Trainable params: 3,011,773 (11.49 MB)

Non-trainable params: 2,415,400 (9.21 MB)

```
W0000 00:00:1742702345.179863   26164 assert_op.cc:38] Ignoring Assert operator compile_loss/sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/assert_equal_1/Assert/Assert
  1/1302 ━━━━━━━━━━━━━━━━━━━━ 3:43:05 10s/step - accuracy: 0.0000e+00 - loss: nan
2025-03-22 21:59:10.919677: I external/local_xla/xla/stream_executor/cuda/cuda_asm_compiler.cc:397] ptxas warning : Registers are spilled to local memory in function 'input_multiply_reduce_fusion', 88 bytes spill stores, 84 bytes spill loads

 120/1302 ━━━━━━━━━━━━━━━━━━━━ 42s 36ms/step - accuracy: 0.6176 - loss: nan
W0000 00:00:1742702355.648230   26162 assert_op.cc:38] Ignoring Assert operator compile_loss/sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/assert_equal_1/Assert/Assert
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.6432 - loss: nan
W0000 00:00:1742702409.246128   26165 assert_op.cc:38] Ignoring Assert operator compile_loss/sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/assert_equal_1/Assert/Assert
W0000 00:00:1742702412.768512   26161 assert_op.cc:38] Ignoring Assert operator compile_loss/sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/assert_equal_1/Assert/Assert
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 74s 49ms/step - accuracy: 0.6432 - loss: nan - val_accuracy: 0.6437 - val_loss: nan
```

Out[23]: <keras.src.callbacks.history.History at 0x7fc4a00ce7d0>

In [24]:
```
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 = transformer1(
            {
                "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(30):
    input_sentence = random.choice(test_eng_texts)
    translated = decode_sequence(input_sentence)
```

In [25]:
```
print(input_sentence)
print(translated)
```

```
They were all friends as children.
[start]
```

Hora de modificar el modelo y cambiar algunas cosas, como el optimizador, las epocas, el n grama o el learning rate

En este caso usamos 30 epoocas, adam optimizer, learning rate de 1e_4, y ngrama de 30

In [26]:
```
strip_chars = string.punctuation + "¿"
strip_chars = strip_chars.replace("[", "")
strip_chars = strip_chars.replace("]", "")

vocab_size = 15000
sequence_length = 30
batch_size = 64


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

In [27]:
```
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)
```

In [28]:
```
voc = eng_vectorization.get_vocabulary()
word_index = dict(zip(voc, range(len(voc))))
```

In [31]:
```
num_tokens = 12077
vocab_size = 12077
embedding_dim = 100
hits = 0
misses = 0
embed_dim = 100

embedding_matrix = np.zeros((num_tokens, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:

        embedding_matrix[i] = embedding_vector
        hits += 1
    else:
        misses += 1
print("Converted %d words (%d misses)" % (hits, misses))
```

```
Converted 11691 words (318 misses)
```

In [32]:
```
from tensorflow.keras.layers import Embedding
encoder_inputs = keras.Input(shape=(None,), dtype="int64", name="encoder_inputs")
x =  Embedding(input_dim=vocab_size, output_dim=embedding_dim,
                           weights=[embedding_matrix], trainable=False)(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_inputs")
x =  Embedding(input_dim=vocab_size, output_dim=embedding_dim,
                           weights=[embedding_matrix], trainable=False)(decoder_inputs)
x = TransformerDecoder(embed_dim, latent_dim, num_heads)([x, encoder_outputs])
decoder_outputs = layers.Dense(vocab_size, activation="softmax")(x)
```

```
decoder = keras.Model([decoder_inputs, encoded_seq_inputs], decoder_outputs)

transformer1 = keras.Model(
    {"encoder_inputs": encoder_inputs, "decoder_inputs": decoder_inputs},
    decoder_outputs,
    name="transformer",
)
```

El modelo desde la primer epoca parece haberse estancado en un accuracy de 75%, y en todas las epocas nunca cambio de accuracy, esto creo qeu puede ser debido al optimizer qeu no sea el adecuado o el learning rate fue demasiado bajo y cayo muy rapido en un optimo del que no pudo salir

In [33]:
```
# Modified training parameters
epochs = 30  # Increased from 1 to 35
learning_rate = 1e-4  # Custom learning rate
optimizer = keras.optimizers.Adam(learning_rate=learning_rate)
transformer1.summary()
transformer1.compile(
    optimizer,
    loss=keras.losses.SparseCategoricalCrossentropy(ignore_class=0),
    metrics=["Accuracy"],
)
transformer1.fit(train_ds, epochs=epochs, validation_data=val_ds)
```

Model: "transformer"

Layer (type)	Output Shape	Param #	Connected to
encoder_inputs (InputLayer)	(None, None)	0	-
decoder_inputs (InputLayer)	(None, None)	0	-
embedding_11 (Embedding)	(None, None, 100)	1,207,700	encoder_inputs[0…
embedding_12 (Embedding)	(None, None, 100)	1,207,700	decoder_inputs[0…
transformer_encode… (TransformerEncode…	(None, None, 100)	734,648	embedding_11[0][…
transformer_decode… (TransformerDecode…	(None, None, 100)	1,057,348	embedding_12[0][… transformer_enco…
dense_18 (Dense)	(None, None, 12077)	1,219,777	transformer_deco…

Total params: 5,427,173 (20.70 MB)

Trainable params: 3,011,773 (11.49 MB)

Non-trainable params: 2,415,400 (9.21 MB)

```
Epoch 1/30
W0000 00:00:1742702557.780413   26165 assert_op.cc:38] Ignoring Assert operator compile_loss/sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/assert_equal_1/Assert/Assert
2025-03-22 22:02:44.941464: I external/local_xla/xla/stream_executor/cuda/cuda_asm_compiler.cc:397] ptxas warning : Registers are spilled to local memory in function 'copy_fusion', 2164 bytes spill stores, 2192 bytes spill loads
ptxas warning : Registers are spilled to local memory in function '__cuda_sm3x_div_rn_noftz_f32_slowpath', 44 bytes spill stores, 44 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'input_multiply_reduce_fusion', 88 bytes spill stores, 84 bytes spill loads
  404/1302 ━━━━━━━━━━━━━━━━━━━━ 45s 51ms/step - Accuracy: 0.7516 - loss: nan
W0000 00:00:1742702586.178946   26161 assert_op.cc:38] Ignoring Assert operator compile_loss/sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/assert_equal_1/Assert/Assert
  405/1302 ━━━━━━━━━━━━━━━━━━━━ 59s 66ms/step - Accuracy: 0.7516 - loss: nan
2025-03-22 22:03:11.886517: I external/local_xla/xla/stream_executor/cuda/cuda_asm_compiler.cc:397] ptxas warning : Registers are spilled to local memory in function 'copy_fusion', 2164 bytes spill stores, 2192 bytes spill loads
ptxas warning : Registers are spilled to local memory in function '__cuda_sm3x_div_rn_noftz_f32_slowpath', 44 bytes spill stores, 44 bytes spill loads
1301/1302 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - Accuracy: 0.7595 - loss: nan
W0000 00:00:1742702638.742208   26160 assert_op.cc:38] Ignoring Assert operator compile_loss/sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/assert_equal_1/Assert/Assert
W0000 00:00:1742702643.398446   26165 assert_op.cc:38] Ignoring Assert operator compile_loss/sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/assert_equal_1/Assert/Assert
```

```
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 98s 64ms/step - Accuracy: 0.7595 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 2/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 53ms/step - Accuracy: 0.7643 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 3/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 71s 54ms/step - Accuracy: 0.7637 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 4/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 71s 54ms/step - Accuracy: 0.7648 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 5/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 71s 54ms/step - Accuracy: 0.7644 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 6/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - Accuracy: 0.7639 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 7/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 71s 54ms/step - Accuracy: 0.7647 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 8/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - Accuracy: 0.7644 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 9/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - Accuracy: 0.7642 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 10/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - Accuracy: 0.7631 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 11/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 53ms/step - Accuracy: 0.7638 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 12/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 53ms/step - Accuracy: 0.7649 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 13/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - Accuracy: 0.7642 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 14/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - Accuracy: 0.7644 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 15/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 69s 53ms/step - Accuracy: 0.7640 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 16/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 84s 54ms/step - Accuracy: 0.7644 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 17/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - Accuracy: 0.7643 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 18/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 69s 53ms/step - Accuracy: 0.7645 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 19/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - Accuracy: 0.7639 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 20/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 71s 55ms/step - Accuracy: 0.7642 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 21/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - Accuracy: 0.7642 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 22/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - Accuracy: 0.7646 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 23/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - Accuracy: 0.7636 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 24/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - Accuracy: 0.7640 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 25/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - Accuracy: 0.7646 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 26/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 71s 54ms/step - Accuracy: 0.7643 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 27/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - Accuracy: 0.7644 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 28/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 71s 54ms/step - Accuracy: 0.7642 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 29/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 72s 55ms/step - Accuracy: 0.7641 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
Epoch 30/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 72s 55ms/step - Accuracy: 0.7647 - loss: nan - val_Accuracy: 0.7622 - val_loss: nan
```

Out[33]: <keras.src.callbacks.history.History at 0x7fc47d2c9d80>

In [34]:
```python
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 = nuevo_modelo(
            {
                "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(30):
    input_sentence = random.choice(test_eng_texts)
    translated = decode_sequence(input_sentence)
```

In [35]:
```python
print(input_sentence)
print(translated)
```

```
She likes miniskirts.
[start] le gusta la gusta [UNK] [end]
```

Conclusiones

Lo realmente difícil para mí fue intentar implementar las gráficas de weight activation, busque documentación en internet, foros y hasta IAs pero ninguna resulto, creo sería muy importante hacer esta clase de Código desde cero en clase ya que nunca se hace Código en clase y el profesor solo ve la "teoría" pero nunca profundiza en nada y con información actualizada ya que el material de la que hay afuera en internet esta desactualizada a las nuevas versiones de TF. Otro punto que se me dificulto es los metrics, realmente ahí ya no tuve tiempo de completarlo, pero no entiendo muy bien con es que se usan o se llaman.

El uso de embeddings acelera en gran medida los cálculos y la convergencia, y otra vez se demuestra que se necesitan de computadoras con un gran poder de cómputo para hacer este tipo de modelos.