English to Spanish Translation with ***Transformers***

This tutorial demonstrates how to create and train a Transformer model to translate Sinhala into English. The Transformer was originally proposed in "Attention is all you need" by Vaswani et al. (2017).

The following animation shows how the transformation works in language translation.

```
from IPython.display import Image

Image(url='https://www.tensorflow.org/images/tutorials/transformer/apply_the_trans
```

Necessary Library Imports

```
import random
import tensorflow as tf
import string
import re
from tensorflow import keras
from tensorflow.keras import layers
```

Prepare the Data

Mount the Google Drive

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

```
    Mounted at /content/drive
```

Read the data file

```
text_file = "/content/drive/My Drive/Colab_Data_Files/spa.txt"
with open(text_file) as f:
    lines = f.read().split("\n")[:-1]

i = 0
for line in lines:
  print(line)
  i = i + 1
  if(i==20):
    break
```

```
    Go.     Ve.
    Go.     Vete.
    Go.     Vaya.
    Go.     Váyase.
    Hi.     Hola.
    Run!    ¡Corre!
    Run.    Corred.
    Who?    ¿Quién?
    Fire!   ¡Fuego!
    Fire!   ¡Incendio!
    Fire!   ¡Disparad!
    Help!   ¡Ayuda!
    Help!   ¡Socorro! ¡Auxilio!
    Help!   ¡Auxilio!
    Jump!   ¡Salta!
    Jump.   Salte.
    Stop!   ¡Parad!
    Stop!   ¡Para!
    Stop!   ¡Pare!
    Wait!   ¡Espera!
```

```
for x in range(len(lines)-10,len(lines)):
  print(lines[x])
```

```
You can't view Flash content on an iPad. However, you can easily email yoursel
A mistake young people often make is to start learning too many languages at t
No matter how much you try to convince people that chocolate is vanilla, it'll
In 1969, Roger Miller recorded a song called "You Don't Want My Love." Today,
A child who is a native speaker usually knows many things about his or her lar
There are four main causes of alcohol-related death. Injury from car accidents
There are mothers and fathers who will lie awake after the children fall aslee
A carbon footprint is the amount of carbon dioxide pollution that we produce a
Since there are usually multiple websites on any given topic, I usually just c
If you want to sound like a native speaker, you must be willing to practice sa
```

Split the English and Spanish translation pairs

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

```
for i in range(3):
  print(random.choice(text_pairs))
```

```
('I thought Tom would take Mary out for dinner.', '[start] Pensé que Tom lleva
("It's a hard question.", '[start] Es una pregunta difícil. [end]')
("I'm being good to you this morning.", '[start] Estoy siendo bueno contigo es
```

Randomize the data

```
import random
random.shuffle(text_pairs)
```

Spliting the data into training, validation and Testing

```
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("Total sentences:",len(text_pairs))
print("Training set size:",len(train_pairs))
print("Validation set size:",len(val_pairs))
print("Testing set size:",len(test_pairs))
```

```
Total sentences: 118964
Training set size: 83276
Validation set size: 17844
Testing set size: 17844
```

```
len(train_pairs)+len(val_pairs)+len(test_pairs)
```

```
118964
```

Removing Punctuations

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

```
f"[{re.escape(strip_chars)}]"
```

```
'[!"\\#\\$%\\&\'\\(\\)\\*\\+,\\-\\./:;<=>\\?@\\\\\\^_`\\{\\|\\}\\~¿]'
```

```
f"{3+5}"
```

```
'8'
```

Vectorizing the English and Spanish text pairs

```
def custom_standardization(input_string):
    lowercase = tf.strings.lower(input_string)
    return tf.strings.regex_replace(
        lowercase, f"[{re.escape(strip_chars)}]", "")

vocab_size = 15000
sequence_length = 20

source_vectorization = layers.TextVectorization(
    max_tokens=vocab_size,
    output_mode="int",
    output_sequence_length=sequence_length,
)
target_vectorization = layers.TextVectorization(
    max_tokens=vocab_size,
    output_mode="int",
    output_sequence_length=sequence_length + 1,
    standardize=custom_standardization,
)
train_english_texts = [pair[0] for pair in train_pairs]
train_spanish_texts = [pair[1] for pair in train_pairs]
```

```
source_vectorization.adapt(train_english_texts)
target_vectorization.adapt(train_spanish_texts)
```

Preparing datasets for the translation task

```
batch_size = 64

def format_dataset(eng, spa):
    eng = source_vectorization(eng)
    spa = target_vectorization(spa)
    return ({
        "english": eng,
        "spanish": 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, num_parallel_calls=4)
    return dataset.shuffle(2048).prefetch(16).cache()

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

```
for inputs, targets in train_ds.take(1):
    print(f"inputs['english'].shape: {inputs['english'].shape}")
    print(f"inputs['spanish'].shape: {inputs['spanish'].shape}")
    print(f"targets.shape: {targets.shape}")
```

```
inputs['english'].shape: (64, 20)
inputs['spanish'].shape: (64, 20)
targets.shape: (64, 20)
```

```
print(list(train_ds.as_numpy_iterator())[50])
```

```
({'english': array([[   5, 234,  39, ...,   0,   0,   0],
       [ 77, 982,  17, ...,   0,   0,   0],
       [ 21,  66,  90, ...,   0,   0,   0],
       ...,
       [177,   3, 129, ...,   0,   0,   0],
       [  3,  60,   9, ...,   0,   0,   0],
       [ 22, 163,   6, ...,   0,   0,   0]]), 'spanish': array([[   2,   28, 2
       [   2, 4279,   10, ...,    0,    0,    0],
       [   2,   26,    7, ...,    0,    0,    0],
       ...,
       [   2,   54, 3192, ...,    0,    0,    0],
       [   2,   84,    5, ...,    0,    0,    0],
       [   2,    7,  862, ...,    0,    0,    0]])}, array([[  28, 2177,    4,
       [4279,   10,  163, ...,    0,    0,    0],
       [  26,    7, 7004, ...,    0,    0,    0],
       ...,
       [  54, 3192,   57, ...,    0,    0,    0],
       [  84,    5,   44, ...,    0,    0,    0],
       [   7,  862,    5, ...,    0,    0,    0]]))
```

Transformer encoder implemented as a subclassed Layer

```
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()

    def call(self, inputs, mask=None):
        if mask is not None:
            mask = mask[:, tf.newaxis, :]
        attention_output = self.attention(
            inputs, inputs, attention_mask=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,
            "num_heads": self.num_heads,
            "dense_dim": self.dense_dim,
        })
        return config
```

The Transformer decoder

```
class TransformerDecoder(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_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(dense_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 get_config(self):
        config = super().get_config()
        config.update({
            "embed_dim": self.embed_dim,
            "num_heads": self.num_heads,
            "dense_dim": self.dense_dim,
        })
        return config

    def get_causal_attention_mask(self, inputs):
        input_shape = tf.shape(inputs)
        batch_size, sequence_length = input_shape[0], input_shape[1]
        i = tf.range(sequence_length)[:, tf.newaxis]
        j = tf.range(sequence_length)
        mask = tf.cast(i >= j, dtype="int32")
        mask = tf.reshape(mask, (1, input_shape[1], input_shape[1]))
        mult = tf.concat(
            [tf.expand_dims(batch_size, -1),
             tf.constant([1, 1], dtype=tf.int32)], axis=0)
        return tf.tile(mask, mult)

    def call(self, inputs, encoder_outputs, mask=None):
        causal_mask = self.get_causal_attention_mask(inputs)
        if mask is not None:
            padding_mask = tf.cast(
                mask[:, tf.newaxis, :], dtype="int32")
            padding_mask = tf.minimum(padding_mask, causal_mask)
        else:
            padding_mask = mask
        attention_output_1 = self.attention_1(
            query=inputs,
            value=inputs,
            key=inputs,
            attention_mask=causal_mask)
        attention_output_1 = self.layernorm_1(inputs + attention_output_1)
        attention_output_2 = self.attention_2(
            query=attention_output_1,
            value=encoder_outputs,
            key=encoder_outputs,
            attention_mask=padding_mask,
        )
        attention_output_2 = self.layernorm_2(
            attention_output_1 + attention_output_2)
        proj_output = self.dense_proj(attention_output_2)
        return self.layernorm_3(attention_output_2 + proj_output)
```

Positional Encoding

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

    def call(self, inputs):
        length = tf.shape(inputs)[-1]
        positions = tf.range(start=0, limit=length, delta=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 tf.math.not_equal(inputs, 0)

    def get_config(self):
        config = super(PositionalEmbedding, self).get_config()
        config.update({
            "output_dim": self.output_dim,
            "sequence_length": self.sequence_length,
            "input_dim": self.input_dim,
        })
        return config
```

End-to-end Transformer

```
embed_dim = 256
dense_dim = 2048
num_heads = 8

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

decoder_inputs = keras.Input(shape=(None,), dtype="int64", name="spanish")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(decoder_inputs)
x = TransformerDecoder(embed_dim, dense_dim, num_heads)(x, encoder_outputs)
x = layers.Dropout(0.5)(x)
decoder_outputs = layers.Dense(vocab_size, activation="softmax")(x)
transformer = keras.Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

```
transformer.summary()
```

```
Model: "model"
________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected
================================================================================
 english (InputLayer)        [(None, None)]               0         []

 spanish (InputLayer)        [(None, None)]               0         []

 positional_embedding (Posi  (None, None, 256)            3845120   ['english[
 tionalEmbedding)

 positional_embedding_1 (Po  (None, None, 256)            3845120   ['spanish[
 sitionalEmbedding)

 transformer_encoder (Trans  (None, None, 256)            3155456   ['position
 formerEncoder)

 transformer_decoder (Trans  (None, None, 256)            5259520   ['position
 formerDecoder)                                                     ',
                                                                     'transfor

 dropout (Dropout)           (None, None, 256)            0         ['transfor

 dense_4 (Dense)             (None, None, 15000)          3855000   ['dropout[

================================================================================
Total params: 19960216 (76.14 MB)
Trainable params: 19960216 (76.14 MB)
Non-trainable params: 0 (0.00 Byte)
________________________________________________________________________________
```

Training the sequence-to-sequence Transformer

```
transformer.compile(
    optimizer="rmsprop",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"])
transformer.fit(train_ds, epochs=30, validation_data=val_ds)
```

```
1302/1302 [==============================] - 88s 67ms/step - loss: 2.8579 - ac
Epoch 3/30
1302/1302 [==============================] - 88s 68ms/step - loss: 2.5606 - ac
Epoch 4/30
1302/1302 [==============================] - 87s 67ms/step - loss: 2.3927 - ac
Epoch 5/30
1302/1302 [==============================] - 87s 67ms/step - loss: 2.2895 - ac
Epoch 6/30
1302/1302 [==============================] - 87s 67ms/step - loss: 2.2148 - ac
Epoch 7/30
1302/1302 [==============================] - 88s 68ms/step - loss: 2.1568 - ac
Epoch 8/30
1302/1302 [==============================] - 88s 67ms/step - loss: 2.0938 - ac
Epoch 9/30
1302/1302 [==============================] - 89s 68ms/step - loss: 2.0357 - ac
```

```
1302/1302 [==============================] - 89s 68ms/step - loss: 2.0357 - ac
Epoch 10/30
1302/1302 [==============================] - 87s 67ms/step - loss: 1.9890 - ac
Epoch 11/30
1302/1302 [==============================] - 88s 68ms/step - loss: 1.9494 - ac
Epoch 12/30
1302/1302 [==============================] - 89s 68ms/step - loss: 1.9171 - ac
Epoch 13/30
1302/1302 [==============================] - 88s 68ms/step - loss: 1.8910 - ac
Epoch 14/30
1302/1302 [==============================] - 89s 68ms/step - loss: 1.8665 - ac
Epoch 15/30
1302/1302 [==============================] - 89s 68ms/step - loss: 1.8484 - ac
Epoch 16/30
1302/1302 [==============================] - 88s 68ms/step - loss: 1.8266 - ac
Epoch 17/30
1302/1302 [==============================] - 90s 69ms/step - loss: 1.8085 - ac
Epoch 18/30
1302/1302 [==============================] - 89s 68ms/step - loss: 1.7907 - ac
Epoch 19/30
1302/1302 [==============================] - 88s 68ms/step - loss: 1.7752 - ac
Epoch 20/30
1302/1302 [==============================] - 88s 68ms/step - loss: 1.7567 - ac
Epoch 21/30
1302/1302 [==============================] - 88s 68ms/step - loss: 1.7421 - ac
Epoch 22/30
1302/1302 [==============================] - 88s 67ms/step - loss: 1.7295 - ac
Epoch 23/30
1302/1302 [==============================] - 88s 68ms/step - loss: 1.7161 - ac
Epoch 24/30
1302/1302 [==============================] - 88s 68ms/step - loss: 1.7021 - ac
Epoch 25/30
1302/1302 [==============================] - 89s 68ms/step - loss: 1.6874 - ac
Epoch 26/30
1302/1302 [==============================] - 88s 68ms/step - loss: 1.6781 - ac
Epoch 27/30
1302/1302 [==============================] - 88s 68ms/step - loss: 1.6642 - ac
Epoch 28/30
1302/1302 [==============================] - 88s 68ms/step - loss: 1.6512 - ac
Epoch 29/30
1302/1302 [==============================] - 88s 68ms/step - loss: 1.6386 - ac
Epoch 30/30
1302/1302 [==============================] - 88s 68ms/step - loss: 1.6250 - ac
<keras.src.callbacks.History at 0x79c860129420>
```

```python
import numpy as np
spa_vocab = target_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 = source_vectorization([input_sentence])
    decoded_sentence = "[start]"
    for i in range(max_decoded_sentence_length):
        tokenized_target_sentence = target_vectorization(
            [decoded_sentence])[:, :-1]
        predictions = transformer(
            [tokenized_input_sentence, tokenized_target_sentence])
        sampled_token_index = np.argmax(predictions[0, i, :])
        sampled_token = spa_index_lookup[sampled_token_index]
        decoded_sentence += " " + sampled_token
        if sampled_token == "[end]":
            break
    return decoded_sentence

test_eng_texts = [pair[0] for pair in test_pairs]
for _ in range(20):
    input_sentence = random.choice(test_eng_texts)
    print("-")
    print(input_sentence)
    print(decode_sequence(input_sentence))
```

```
-
Tom wants a new car.
[start] tom quiere un coche nuevo [end]
-
Are you absolutely sure you want to sell your father's guitar?
[start] estás seguro de que querés abrir tu padre [end]
-
Tom should call a lawyer.
[start] tom debería llamar a un abogado [end]
-
Tom is a troublemaker.
[start] tom es un [UNK] [end]
-
I'm going to go now.
[start] voy a ir a mí en este momento [end]
-
Visitors are requested not to touch the exhibits.
[start] los se les dice que no me [UNK] los has se pueden tocar las cosas [enc
-
Being very tired, I went to bed early.
[start] estar muy cansado me fui temprano a la cama [end]
-
Some people have no patience.
[start] algunas personas no tienes paciencia [end]
-
Painting our house took longer than we expected.
[start] pintura nuestro casa tomó más tiempo de lo que estábamos [end]
-
None of these are mine.
[start] ninguna de estos son mía [end]
-
If he had not died so young, he would have become a great scientist.
[start] si no hubiera oído él no hubiera sido un gran fuerte se enamoró de gra
-
Don't try to fool me.
[start] no me [UNK] [end]
-
You're getting closer.
[start] estás cada vez [end]
-
I'm not interested in your opinion.
[start] no me dan interesado en tu opinión [end]

-
Tom isn't watching a basketball game on TV.
[start] tom no está haciendo un juego de tenis de televisión [end]
```

```
-
I believe it is a genuine Picasso.
[start] creo que es un [UNK] como un lago más profundo [end]
-
Someday I'll run like the wind.
[start] algún día me [UNK] el tiempo [end]
-
Tom is going to need help.
[start] tom se va a conseguir ayuda [end]
-
Do you know Tom personally?
[start] conoce a tom también a los platos [end]
```