Diplomado Inteligencia Artificial y Ciencia de Datos: Métodos fundamentales, Progra Lógica, Ciencia de Computación e Ingeniería de Computación

Deep Learning: Transformers Challenge

Carolina Bernal Rodríguez
Sati March, 22nd 2025

OVERVIEW

Use the code https://keras.io/examples/nlp/neural_machine_translation_with_transformer/

1. Make it run. Note that it uses the spa-eng file from the Anki site, Include code to save the model on disk so that you can use the pre-trained model.
2. Include code to use the pre-trained embeddings from Stanford. Link at https://nlp.stanford.edu/projects/glove/.
3. Include code to show the layer Code at https://github.com/tensorflow/text/blob/master/docs/tutorials/transformer.ipynb
4. Work with the model to improve its performance. Things to try are:
    a. Use more than 30 epochs
    b. Change the number of ngrams
    c. Change the learning rate
    d. Change the optimizer
    e. Change the metric
    f. Explore how to use the BLUE (Bilingual Evaluation Understudy)
    g. Explore how to use the Rouge score

Code step by step

The first step is the usual for all challenges, we need to get the data, in this case we'll be working with an English-to-Spanish translation dataset provided by Anki. Let's download it:

```
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_extracted" /
"spa-eng" / "spa.txt"
```

The next step should be parsing the data, we know the text structure consists of pairs or English - Spanish sentences (translation). The English sentence is the source sequence and the Spanish one is the target sequence. We prepend the token "[start]" and we append the token "[end]" to the Spanish sentence.

The next step is also expected when you want to train a model, you'll need to create your testing and validation sets.

```
for _ in range(5):

print(random.choice(text_pairs))
```

```
('Good work, Tom.', '[start] Buen trabajo, Tom. [end]') ('They got to be good friends.', '[start] Llegaron a ser buenos amigos. [end]') ('The population has doubled in the last five years.', '[start] La población se ha duplicado en los últimos cinco años. [end]') ('I keep in touch with Tom.', '[start] Lo veo a Tomás de vez en cuando. [end]') ('I need to see you in my office.', '[start] Necesito verte en mi oficina. [end]')
```

```
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
```

The next part is also important since we will need to vectorize de text data, to achieve that we'll use two instances of the TextVectorization layer to vectorize the text data (one for English and one for Spanish), that is to say, to turn the original strings into integer sequences where each integer represents the index of a word in a vocabulary. Then we can form our datasets.

```
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_f = make_dataset(train_pairs)
```

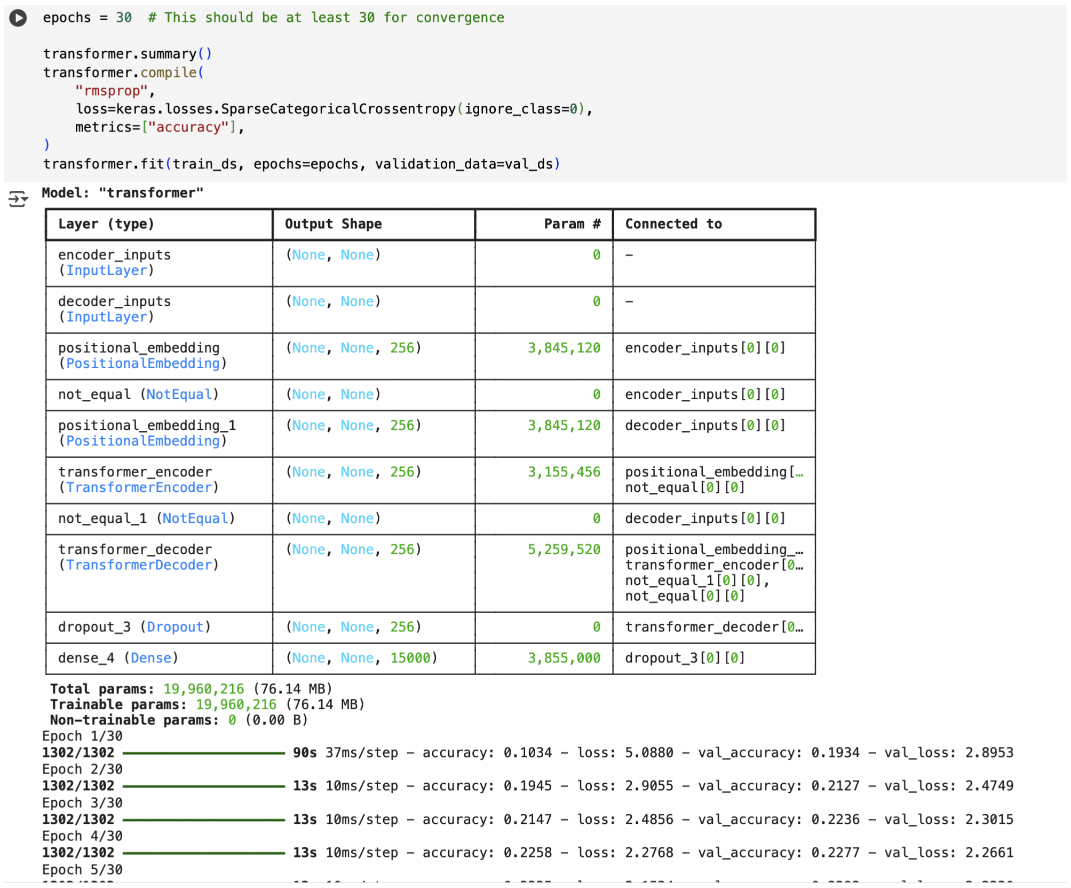
```
val_ds_f = make_dataset(val_pairs)
test_ds_f = make_dataset(test_pairs)
```

The most important step is to build our model Transformer which consists of a **TransformerEncoder** and a **TransformerDecoder** chained together. To make the model aware of word order, we also use a **PositionalEmbedding** layer.

A key detail that makes this possible is causal masking (see method get_causal_attention_mask() on the TransformerDecoder). The TransformerDecoder sees the entire sequences at once, and thus we must make sure that it only uses information from target tokens 0 to N when predicting token N+1 (otherwise, it could use information from the future, which would result in a model that cannot be used at inference time).

Another important part should be the model assemble end-to-end and defining the model like:

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

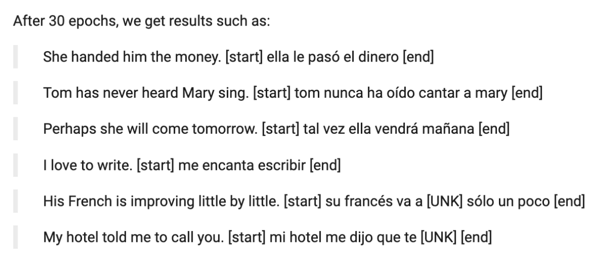


```
epochs = 30  # This should be at least 30 for convergence

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             │ (None, None)            │              0 │ -                      │
│ (InputLayer)               │                         │                │                        │
├────────────────────────────┼─────────────────────────┼────────────────┼────────────────────────┤
│ decoder_inputs             │ (None, None)            │              0 │ -                      │
│ (InputLayer)               │                         │                │                        │
├────────────────────────────┼─────────────────────────┼────────────────┼────────────────────────┤
│ positional_embedding       │ (None, None, 256)       │      3,845,120 │ encoder_inputs[0][0]   │
│ (PositionalEmbedding)      │                         │                │                        │
├────────────────────────────┼─────────────────────────┼────────────────┼────────────────────────┤
│ not_equal (NotEqual)       │ (None, None)            │              0 │ encoder_inputs[0][0]   │
├────────────────────────────┼─────────────────────────┼────────────────┼────────────────────────┤
│ positional_embedding_1     │ (None, None, 256)       │      3,845,120 │ decoder_inputs[0][0]   │
│ (PositionalEmbedding)      │                         │                │                        │
├────────────────────────────┼─────────────────────────┼────────────────┼────────────────────────┤
│ transformer_encoder        │ (None, None, 256)       │      3,155,456 │ positional_embedding[… │
│ (TransformerEncoder)       │                         │                │ not_equal[0][0]        │
├────────────────────────────┼─────────────────────────┼────────────────┼────────────────────────┤
│ not_equal_1 (NotEqual)     │ (None, None)            │              0 │ decoder_inputs[0][0]   │
├────────────────────────────┼─────────────────────────┼────────────────┼────────────────────────┤
│ transformer_decoder        │ (None, None, 256)       │      5,259,520 │ positional_embedding_… │
│ (TransformerDecoder)       │                         │                │ transformer_encoder[0… │
│                            │                         │                │ not_equal_1[0][0],     │
│                            │                         │                │ not_equal[0][0]        │
├────────────────────────────┼─────────────────────────┼────────────────┼────────────────────────┤
│ dropout_3 (Dropout)        │ (None, None, 256)       │              0 │ transformer_decoder[0… │
├────────────────────────────┼─────────────────────────┼────────────────┼────────────────────────┤
│ 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)
Epoch 1/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 90s 37ms/step - accuracy: 0.1034 - loss: 5.0880 - val_accuracy: 0.1934 - val_loss: 2.8953
Epoch 2/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 13s 10ms/step - accuracy: 0.1945 - loss: 2.9055 - val_accuracy: 0.2127 - val_loss: 2.4749
Epoch 3/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 13s 10ms/step - accuracy: 0.2147 - loss: 2.4856 - val_accuracy: 0.2236 - val_loss: 2.3015
Epoch 4/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 13s 10ms/step - accuracy: 0.2258 - loss: 2.2768 - val_accuracy: 0.2277 - val_loss: 2.2661
Epoch 5/30
Epoch 30/30 1302/1302 ━━━━━━━━━━━━━━━━━━━━ 13s 10ms/step -
accuracy: 0.2743 - loss: 1.5453 - val_accuracy: 0.2333 - val_loss: 2.6468
```

Finally we can decode our text sentences and get results like:



After 30 epochs, we get results such as:

> She handed him the money. [start] ella le pasó el dinero [end]

> Tom has never heard Mary sing. [start] tom nunca ha oído cantar a mary [end]

> Perhaps she will come tomorrow. [start] tal vez ella vendrá mañana [end]

> I love to write. [start] me encanta escribir [end]

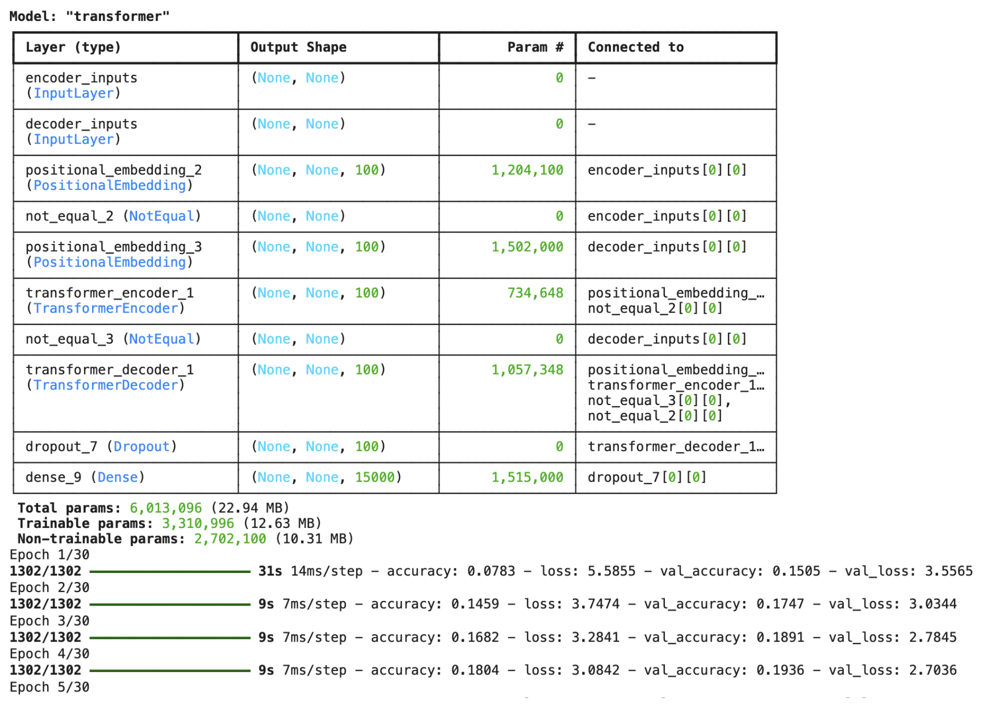
> His French is improving little by little. [start] su francés va a [UNK] sólo un poco [end]

> My hotel told me to call you. [start] mi hotel me dijo que te [UNK] [end]

We can also get better results when the data has better quality (since it's in spanish, accented words cause issues and worsen the training), when adding:

```python
import unicodedata
def remove_accented_char(texto):
# Normalizar el texto a la forma NFD
texto = unicodedata.normalize("NFD", texto)
# Reemplazar los caracteres diacríticos, pero dejando la "ñ" intacta
texto = re.sub(r"(?<!n)[\u0300-\u036f]", "", texto)
# Volver a la forma NFC para evitar problemas de codificación
return unicodedata.normalize("NFC", texto)
```

And re-running the previous steps we



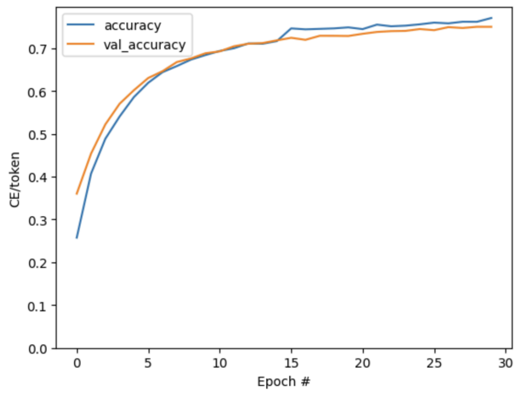
```
Model: "transformer"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                ┃ Output Shape             ┃        Param # ┃ Connected to               ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
│ encoder_inputs              │ (None, None)             │              0 │ -                          │
│ (InputLayer)                │                          │                │                            │
├─────────────────────────────┼──────────────────────────┼────────────────┼────────────────────────────┤
│ decoder_inputs              │ (None, None)             │              0 │ -                          │
│ (InputLayer)                │                          │                │                            │
├─────────────────────────────┼──────────────────────────┼────────────────┼────────────────────────────┤
│ positional_embedding_2      │ (None, None, 100)        │      1,204,100 │ encoder_inputs[0][0]       │
│ (PositionalEmbedding)       │                          │                │                            │
├─────────────────────────────┼──────────────────────────┼────────────────┼────────────────────────────┤
│ not_equal_2 (NotEqual)      │ (None, None)             │              0 │ encoder_inputs[0][0]       │
├─────────────────────────────┼──────────────────────────┼────────────────┼────────────────────────────┤
│ positional_embedding_3      │ (None, None, 100)        │      1,502,000 │ decoder_inputs[0][0]       │
│ (PositionalEmbedding)       │                          │                │                            │
├─────────────────────────────┼──────────────────────────┼────────────────┼────────────────────────────┤
│ transformer_encoder_1       │ (None, None, 100)        │        734,648 │ positional_embedding_…     │
│ (TransformerEncoder)        │                          │                │ not_equal_2[0][0]          │
├─────────────────────────────┼──────────────────────────┼────────────────┼────────────────────────────┤
│ not_equal_3 (NotEqual)      │ (None, None)             │              0 │ decoder_inputs[0][0]       │
├─────────────────────────────┼──────────────────────────┼────────────────┼────────────────────────────┤
│ transformer_decoder_1       │ (None, None, 100)        │      1,057,348 │ positional_embedding_…     │
│ (TransformerDecoder)        │                          │                │ transformer_encoder_1…     │
│                             │                          │                │ not_equal_3[0][0],         │
│                             │                          │                │ not_equal_2[0][0]          │
├─────────────────────────────┼──────────────────────────┼────────────────┼────────────────────────────┤
│ dropout_7 (Dropout)         │ (None, None, 100)        │              0 │ transformer_decoder_1…     │
├─────────────────────────────┼──────────────────────────┼────────────────┼────────────────────────────┤
│ dense_9 (Dense)             │ (None, None, 15000)      │      1,515,000 │ dropout_7[0][0]            │
└─────────────────────────────┴──────────────────────────┴────────────────┴────────────────────────────┘
 Total params: 6,013,096 (22.94 MB)
 Trainable params: 3,310,996 (12.63 MB)
 Non-trainable params: 2,702,100 (10.31 MB)
Epoch 1/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 31s 14ms/step - accuracy: 0.0783 - loss: 5.5855 - val_accuracy: 0.1505 - val_loss: 3.5565
Epoch 2/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - accuracy: 0.1459 - loss: 3.7474 - val_accuracy: 0.1747 - val_loss: 3.0344
Epoch 3/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - accuracy: 0.1682 - loss: 3.2841 - val_accuracy: 0.1891 - val_loss: 2.7845
Epoch 4/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - accuracy: 0.1804 - loss: 3.0842 - val_accuracy: 0.1936 - val_loss: 2.7036
Epoch 5/30
```

saw an small increase on the accuracy (even with not all the trainable params):

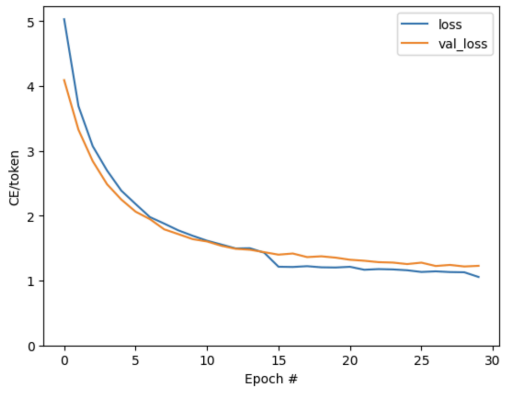
```
Epoch 30/30 1302/1302 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step -
accuracy: 0.2254 - loss: 2.3826 - val_accuracy: 0.2155 - val_loss: 2.5356
```

For the model, re-dined for ex 4, the curves and accuracy are better as shown on the tables:

```
<matplotlib.legend.Legend at 0x7d6a7e089490>
```

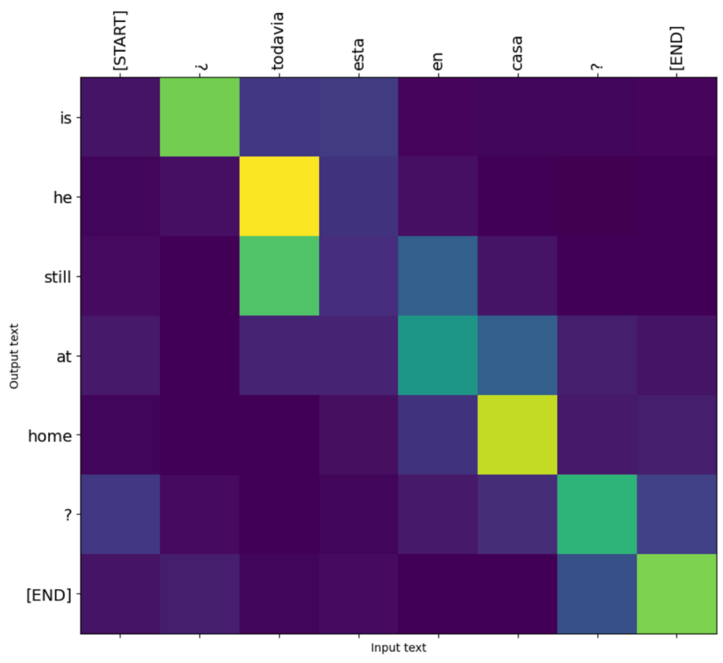


```
<matplotlib.legend.Legend at 0x7d6b6d2e1b90>
```

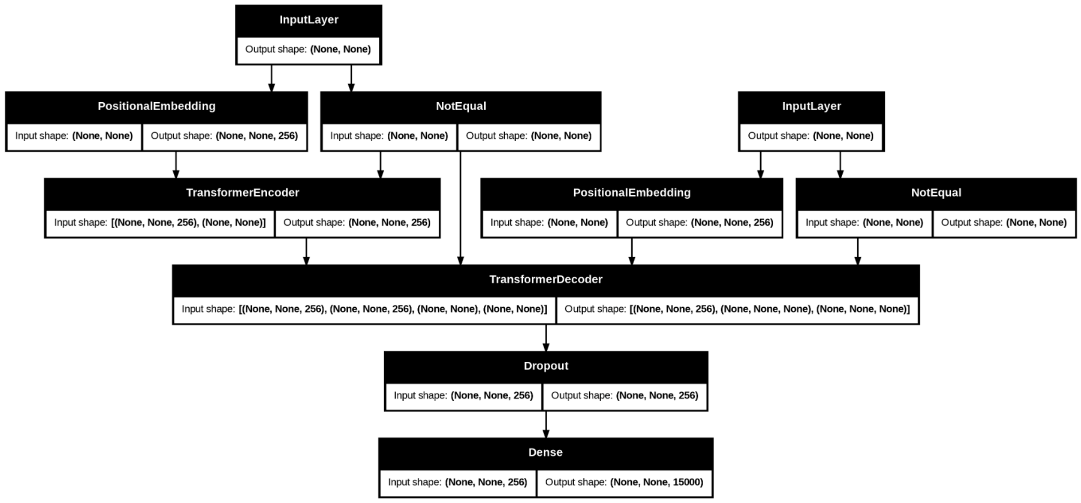


And for the attention plots_

```
<ipython-input-47-93059eaa9bfc>:23: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks() or using a FixedLocator.
  ax.set_xticklabels([''] + context, fontdict=fontdict, rotation=90)
<ipython-input-47-93059eaa9bfc>:24: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks() or using a FixedLocator.
  ax.set_yticklabels([''] + output, fontdict=fontdict)
```



```
from tensorflow.keras.utils import plot_model
plot_model(transformer, to_file='transformer.png', show_shapes=True)
from IPython.display import Image
Image("transformer.png")
```


InputLayer
Output shape: (None, None)
PositionalEmbedding
Input shape: (None, None)
Output shape: (None, None, 256)
NotEqual
Input shape: (None, None)
Output shape: (None, None)
InputLayer
Output shape: (None, None)
TransformerEncoder
Input shape: [(None, None, 256), (None, None)]
Output shape: (None, None, 256)
PositionalEmbedding
Input shape: (None, None)
Output shape: (None, None, 256)
NotEqual
Input shape: (None, None)
Output shape: (None, None)
TransformerDecoder
Input shape: [(None, None, 256), (None, None, 256), (None, None), (None, None)]
Output shape: [(None, None, 256), (None, None, None), (None, None, None)]
Dropout
Input shape: (None, None, 256)
Output shape: (None, None, 256)
Dense
Input shape: (None, None, 256)
Output shape: (None, None, 15000)

Conclusions and Learnings

➔ It was challenging to understand the model , understanding the Encoder and Decoder layers and how it can be used for sequence-to-sequence learnings. On how it can be resumed as the encoder processes an input sequence to then produce the set of context vectors then used by the decoder to generate the output sequence. And how the encoder principal component should be the self-attention mechanism which determines token weights and reflects on inter-token relationships. The Decoder is similar to the encoder but also incl

➔ When plotting the activation layer, the version of tensorflow-text of the original notebook should be modified from 2.18 ➔ 2.15 to successfully run the notebook

➔ The accuracy is better if the input source text and the output target text is clean.