Transformer Challenge

Fausto Morales
ffmogbaj@gmail.com

1. Trabajo realizado

Se construyó el notebook: Transformer Challenge.ipynb donde:

Se ejecutó el código de la siguiente url: English-to-Spanish translation with a sequence-to-sequence Transformer

Paso 1.- Se importaron las librerías de Python a emplearse

Paso 2.- Se importó el set de datos desde: Google APIS con los datos de traducciones de español a inglés.

Paso 3.- Se leyeron los datos para obtener un arreglo del siguiente formato:

```
('I like the one with a white belt.',
'[start] Me gusta el del cinturón
blanco. [end]')
```

Paso 4.- Posteriormente se separó el código en sets de entrenamiento, validación y testing.

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

Paso 5.- Se definieron parámetros para realizar el vocabulario:

```
vocab_size = 5000
sequence_length = 25
batch_size = 64
```

También se realizan funciones para poder eliminar caracteres especiales y volver todo a minúsculas de forma estandarizada. Así como se generan funciones de vectorización a través del vocabulario generando "tokens" por cada palabra bajo las funciones:

```
eng_vectorization
spa_vectorization
```

Paso 6.- Se prepara el texto para realizar un conjunto de datos de entrenamiento y validación a partir de pares de texto en inglés y español. Estos datos en particular tienen la característica que son una tupla con las entradas del codificador (token en inglés) y el decodificador (token en español), y la secuencia esperada del decodificador (token de texto en español desplazado una posición).

Paso 7.- Con el paso 6, se generan tensores, se almacenan en caché, se ordenan de forma aleatoria y se preprocesan para formar dos conjuntos de datos: uno para entrenamiento (train_ds0) y otro para validación (val_ds0).

El tensor train_ds0 tiene las siguientes propiedades:

```
inputs["encoder_inputs"].shape:
(64, 25)
inputs["decoder_inputs"].shape:
(64, 25)
targets.shape: (64, 25)
```

Paso 8.- Se definen tres clases que implementan los componentes principales de un modelo de Transformer:

TransformerEncoder: Implementa el codificador de un Transformer que procesa las entradas utilizando atención multi-cabeza, seguido de una proyección densa.

PositionalEmbedding: Agrega información posicional a las representaciones de los tokens para que el modelo pueda tener en cuenta la posición de los tokens en la secuencia.

TransformerDecoder: Implementa el decodificador de un Transformer que usa atención causal y atención entre el codificador y el decodificador para generar la salida.

Paso 9.- Se define un modelo que usa estas clases.

Layer (type)	Output Shape	Param #	Connected to
encoder_inputs (InputLayer)	(None, None)	0	–
decoder_inputs (InputLayer)	(None, None)	0	–
positional_embeddi… (PositionalEmbeddi…	(None, None, 128)	633,150	encoder_inputs[0…
not_equal_2 (NotEqual)	(None, None)	0	encoder_inputs[0…
positional_embeddi… (PositionalEmbeddi…	(None, None, 128)	633,150	decoder_inputs[0…
transformer_encode… (TransformerEncode…	(None, None, 128)	776,956	positional_embed… not_equal_2[0][0]
not_equal_3 (NotEqual)	(None, None)	0	decoder_inputs[0…
transformer_decode… (TransformerDecode…	(None, None, 128)	1,288,390	positional_embed… transformer_enco… not_equal_3[0][0… not_equal_2[0][0]
dropout_11 (Dropout)	(None, None, 128)	0	transformer_deco…
dense_12 (Dense)	(None, None, 5000)	635,000	dropout_11[0][0]

```
Total params: 3,966,646 (15.13 MB)
```

Optimizador empleado:

```
rmsprop
```

Loss:

```
SparseCategoricalCrossentropy
```

Metrica:

```
accuracy
```

Paso 10.- Se entrenó el modelo con 15 epocas.

En el último step los resultados fueron los siguientes:

```
Epoch 15/15
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 199s
153ms/step - accuracy: 0.1201 - loss:
3.6675 - val_accuracy: 0.1218 -
val_loss: 3.6412
```

Como vemos la precisión es sólo del 12% y el tiempo implicado fue de ~153 ms/paso.

Donde el tiempo de ejecución fue demasiado. Se ejecutaron 15 épocas donde cada una tiene 1,302 pasos que usan ~150 ms/paso. Tenemos:

$$15\ epoca * 1{,}302 \frac{pasos}{epoca} * 150 \frac{ms}{paso} = 2.93M\ ms \sim 49\ min.$$

Se guardan las cosas en "history" como "model original" para compararlo con las mejoras al modelo y se intentan traducir.

El siguiente reto fue:

- Construir un modelo más optimizado.
- Incluir el embedding pre-entrenado.
- Mostrar gráficos con las capas de atención.
- Usar más de 30 épocas para entrenar.
- Guardar el modelo.
- Cambiar Optimizador, métrica y pérdida
- Generar métricas de Rouge y Blue

Para poder realizarlo se realizó una investigación y se encontró una guía buena pero que tenía errores: Neural machine translation with attention y también se buscó: Using pre-trained word embeddings para poder aplicar embeddings pre entrenados.

Se construyó un nuevo modelo tomando la referencia y se modificó para cumplir lo requerido para el ejercicio.

Paso 0: Se generó un nuevo paso donde se construye la función:

```
ShapeChecker
```

Para poder se usa para verificar que las dimensiones de un tensor sean consistentes durante las operaciones de procesamiento de datos. Esta función aplico en otras capas.

Para este modelo optimizado generamos los mismos pasos del: 1 al 7. Con la diferencia de que:

En este modelo generamos primero los tensores con datos de entrenamiento y validación con el par de fraces inglés y español:

```
tf.Tensor(
[b'I banged my elbow against the
wall.'], shape=(1,), dtype=string)

tf.Tensor(
[b'Me pegu\xc3\xa9 en el codo con la
pared.'], shape=(1,), dtype=string)
```

Dado que se procesaron los primeros 10,000 palabras es posible construir el vocabulario desde 1 sólo batch (8 segundos) en luigar de toda la data (10 minutos), ahorrando tiempo de procesamiento.

Al final se general los datashets en forma de tensores:

```
train_ds
val_ds
```

Posteriormente se transforman en valores estandarizados a través de la función:

```
f_lower_and_split_punct(text)
```

Que a su vez genera los tokens:

```
[START]
[END]
```

Generando la siguiente salida ejemplo:

```
[START] ¿ como estas ? [END]
```

Análogo al modelo original.

Otra diferencia implementada es que el vocabulario se genera a través del layer de Keras: TextVectorization, generando:

```
context_text_processor
```

Para el idioma origen: Inglés, que permite generar vocabulario y tokens de una oración.,

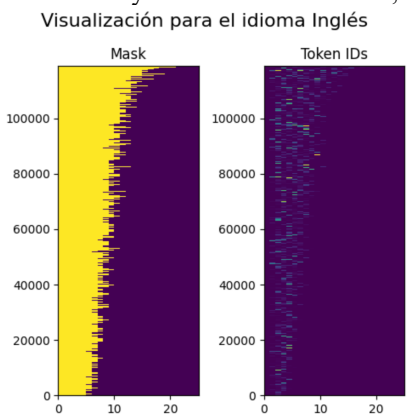


Fig 1. En el "mask" se muestran los datos con valor en amarillo y en cero en morado y para

“Token IDs” los valores del token en un mapa de colores para el idioma Inglés.

Y:

```
target_text_processor
```

Para el idioma objetivo: Español a traducir y que permite generar vocabulario y tokens de una oración.

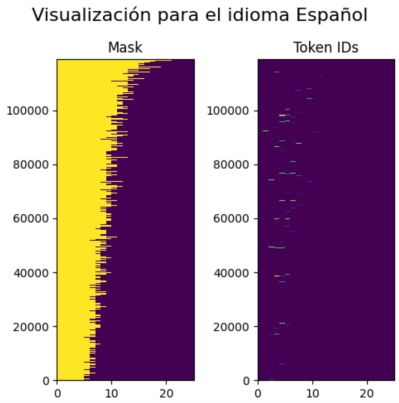


Fig 2. En el “mask” se muestran los datos con valor en amarillo y en cero en morado y para “Token IDs” los valores del token en un mapa de colores para el idioma Español.

En el Paso 8.- Descargamos localmente los embeddings pre-entrenados de la liga: glove de 6B en la misma carpeta donde se encuentra: Transformer Challenge.ipynb donde se encuentran:

```
Found 400000 word vectors.
```

Dado que los embeddings son en inglés; comparamos con el vocabulario construido en inglés y obtenemos:

```
Converted 4349 words (5651 misses)
```

Construyendo la:

```
embedding_layer
```

Esta capa tiene dimensión de 100 vectores.

Continuando la construcción de la arquitectura Transformer:

Generamos la clase Encoder:

El Encoder en este caso tiene:
Un procesamiento de texto inicial que convierte las palabras en vectores y se incluye la importación de los embeddings.
Además, como diferencia con el modelo anterior, empleamos una capa GRU bidireccional que procesa las secuencias de embeddings.

Se define la clase Cross-Attention la cual toma dos entradas: x: La secuencia de consulta (query). context: La secuencia de contexto (value).

La capa anterior aplica el layer MultiHeadAttention donde calcula las puntuaciones de atención y se promedian. Así como se usa el layer: Add que suma las salidas de la atención con la entrada original y el layer: LayerNormalization para estabilizar los gradientes y acelerar el entrenamiento.

Se continia con la definición de la clase Decoder: para procesar las secuencias de entrada (contexto) y generar las predicciones de salida. En este código se realiza la búsqueda de embeddings para la secuencia de salida y genera procesamiento de la secuencia objetivo como una red neuronal recurrente. Posteriormente utiliza una capa de atención sobre la secuencia de contexto y genera predicciones a partir de un output_layer.

Por último, se define la clase Translator, donde se definen los procesadores de texto para transformar las palabras de entrada y salida en tokens (vectores). También se define usar las clases Encoder y Decoder previamente usadas para predecir la traducción.

Cabe mencionar que para para probar la implementación de embeddings pre entrenados se definió 3 calses de Translator distintos:

- Modelo optimizado 1: El primer modelo no incluía embbedings. Sólo una pequeña referencia en el Encoder, sin embargo, posteriormente se hacía referencia al embbeding original del modelo base.
- Modelo optimizado 2: El segundo incluía un embbeding en el Translator pero no se usaba en su función Call, y usaba el mismo Encoder.
- Modelo optimizado 3: El tercero se cambió el Encoder para que las dimensiones fueran de 100 y se usara el embedding con los weights dada la matriz de pesos importada.

Paso 9.-
Se modificaron la clase Translator para poder añadir las siguientes funciones:

```
translate
plot_attention
```

Donde:
La función translate permite que el modelo realice traducciones con un input array con texto.
Mientras la función plot_attention genera y muestra una visualización de la matriz de atención, lo que permite ver cómo cada token en la secuencia de salida está relacionado con los tokens de entrada.

Paso 10.-
Para estos modelos optimizados se cambió el Optimizador a

```
adam
```

Se modificó la metrica y pérdida para:
Generar funciones que solo consideren las posiciones válidas en las secuencias.

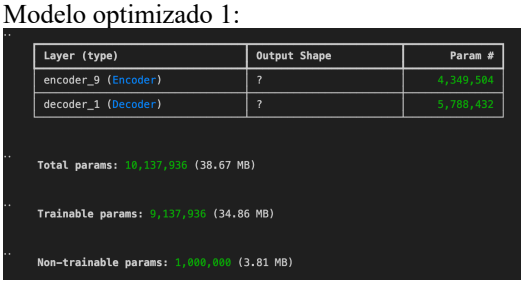
```
masked_acc, masked_loss
```

Ambas se emplea el:

```
SparseCategoricalCrossentropy
```

Y se configuraron 30 épocas en conjunto con un early stop.

Modelo optimizado 1:



Layer (type)	Output Shape	Param #
encoder_9 (Encoder)	?	4,349,504
decoder_1 (Decoder)	?	5,788,432

Total params: 10,137,936 (38.67 MB)

Trainable params: 9,137,936 (34.86 MB)

Non-trainable params: 1,000,000 (3.81 MB)

Se frenó en la época 19:

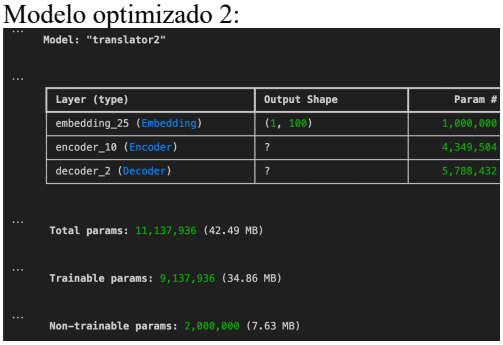
```
Epoch 19/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 29s

292ms/step - loss: 1.2998 -
masked_acc: 0.7249 - masked_loss:
1.2998 - val_loss: 1.4904 -
val_masked_acc: 0.7004 -
val_masked_loss: 1.4904
```

Tiempo estimado:

$$19\ epoca * 100\frac{pasos}{epoca} * 290\frac{ms}{paso} = 511\ K\ ms \sim 9\ min.$$

Modelo optimizado 2:



Model: "translator2"

Layer (type)	Output Shape	Param #
embedding_25 (Embedding)	(1, 100)	1,000,000
encoder_10 (Encoder)	?	4,349,504
decoder_2 (Decoder)	?	5,788,432

Total params: 11,137,936 (42.49 MB)

Trainable params: 9,137,936 (34.86 MB)

Non-trainable params: 2,000,000 (7.63 MB)

Se frenó en la época 30:

```
Epoch 30/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 33s
332ms/step - loss: 1.1785 -
masked_acc: 0.7418 - masked_loss:
1.1791 - val_loss: 1.3154 -
val_masked_acc: 0.7274 -
val_masked_loss: 1.3154
```

Tiempo estimado:

$$30\ epoca * 100\frac{pasos}{epoca} * 320\frac{ms}{paso} = 960\ K\ ms \sim 16\ min.$$

Modelo optimizado 3:

Layer (type)	Output Shape	Param #
encoder2_1 (Encoder2)	?	1,121,200
decoder_1 (Decoder)	?	2,111,200

Total params: 3,232,400 (12.33 MB)

Trainable params: 2,232,400 (8.52 MB)

Non-trainable params: 1,000,000 (3.81 MB)

Se frenó en la época 33:

```
poch 33/40
100/100 ━━━━━━━━━━━━━━━━━━━━ 21s
213ms/step - loss: 1.5312 -
masked_acc: 0.6764 - masked_loss:
1.5312 - val_loss: 1.7743 -
val_masked_acc: 0.6533 -
val_masked_loss: 1.7743
```

Tiempo estimado:

$$33\ epoca * 100\frac{pasos}{epoca} * 210\frac{ms}{paso} = 693\ K\ ms \sim 11\ min.$$

Paso 11.- Graficar History para comparar valores.

Paso 12.- Graficar algunos gráficos de atención.

Paso 13.- Guardar los modelos con la función:

```
tf.saved_model.save
```

realizando la firma de la función

```
translate
```

a partir de la clase

```
Export
```

Paso 14.- Realizar las funciones de métricas Rouge y Blue a través de las paqueterías:

```
keras_nlp.metrics.RougeN
keras_nlp.metrics.Bleu
```

2. Resultados

Según los gráficos para el módulo 2 vemos que el datashet de traducciones: tiene entre 5 a 15 tokens en su mayoría, tanto para inglés como español, por lo que elegir un límite de 25 tokens por traducción es una buena opción.

Ejemplo de traducciones con el modelo original:

```
The balloon is filled with air. :
[start] de de [UNK] [UNK] [UNK] [end]
```

Lo cual fue pésimo.

Ejemplo de traducciones con el modelo 2:

```
'Are  you there?'
```

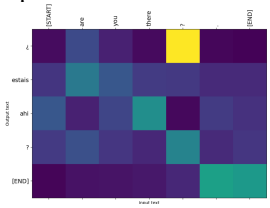
'¿ estais ahi ? '

Por lo que los modelos optimizados son mejores y llegan a presentar datos más certeros.

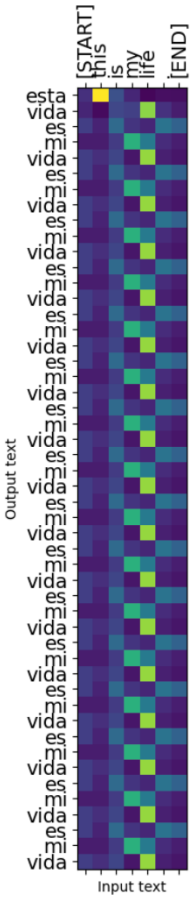
Sin embargo también tienen problemas como loops infinitos.

Esto se puede visualizar en los mapas de atención:

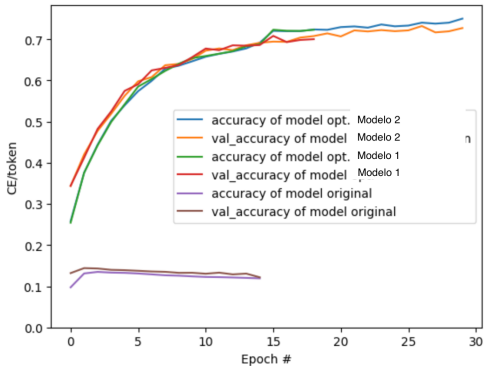
- Ejemplo de traducción correcta:



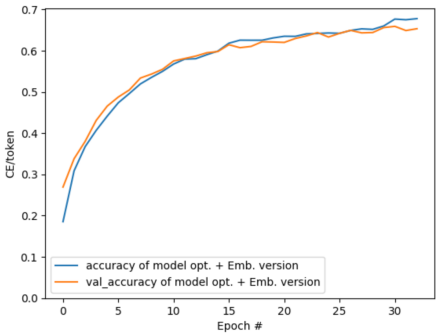
- Ejemplo de traducción con loop:



Al comparar los modelos no optimizados, optimizado 2 y 3; tenemos los siguientes gráficos de accuracy:



El grafico del modelo 3 es el siguiente:



Scores:
Modelo optimizado 1:

```
 Blue score, order 1 = 1.0
 Blue score, order 2 = 0.866
Rouge score, order 1 = 0
Rouge score, order 2 = 0.0792
```

Modelo optimizado 2:

```
 Blue score, order 1 = 1.0
 Blue score, order 2 = 0.866
Rouge score, order 1 = 0.256
Rouge score, order 2 = 0.111
```

Modelo optimizado 3:

```
Blue score, order 1 = 0.0399
Blue score, order 2 = 0.0
Rouge score, order 1= 0.145
Rouge score, order 2= 0.0674
```

Por 30 Frases evaluadas.

3. Conclusiones y comentarios

La traducción y tiempos de procesamiento mejoran muchísimo al emplear un modelo de Red Neuronal Recurrente y simplificar las capas de la red.

El uso de Embeddings en este tipo de red parece no afectar demasiado su rendimiento, para este caso todos convergen alrededor de 40 épocas o menos. El ganador dentro de los 3 modelos optimos fue aquel que no hace explicita su dependencia con Embeddings pre entrenados.

Existen retos que no logré resolver cómo evitar que el modelo no entre en bucle.

Esta vez fue mi primera vez realizando este tipo de ejercicios y métricas.

4. Agredecimientos

Agradezco a Enrique Gomez Cruz por sus ideas para poder resolver este ejercicio.