Transformers Challenge

Implementación y Optimización de un Transformer para Traducción Inglés-Español

1. Introducción

En el campo del Procesamiento del Lenguaje Natural (NLP), la traducción automática ha sido un desafío clave durante décadas. Los primeros enfoques se basaban en reglas lingüísticas, seguidos por modelos estadísticos. Sin embargo, con la llegada del aprendizaje profundo, los modelos basados en redes neuronales han revolucionado la calidad de las traducciones.

Este proyecto se centra en la implementación de un Transformer, un modelo que ha demostrado ser altamente eficiente para tareas de traducción, superando a los enfoques tradicionales como LSTMs y modelos secuencia a secuencia con atención. Nos basamos en el cuaderno disponible en Keras y realizamos mejoras clave para optimizar su rendimiento.

2. Objetivos del Proyecto

Este trabajo tiene como propósito:

1. Obtener el notebook del Neural Machine Translation with Transformer
2. Ejecutar un modelo Transformer para la traducción de inglés a español, utilizando el conjunto de datos spa-eng de Anki.

Primero importamos las bibliotecas necesarias para el proyecto, incluyendo TensorFlow, Keras y otras herramientas para el procesamiento de texto y datos. También configura el backend de Keras a TensorFlow.
Posteriormente descargamos el conjunto de datos de traducción inglés-español de Anki y lo extrae.
Se lee el archivo de texto, divide las líneas en pares de frases inglés-español, y agrega los tokens "[start]" y "[end]" a las frases en español, para posteriormente repetir el proceso de la celda anterior, leyendo y procesando el archivo para obtener los pares de frases

```
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))
```

```
# Now, read and process the file
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))
```

Dividimos el conjunto de datos en conjuntos de entrenamiento, validación y prueba, utilizando un 15% para validación y otro 15% para prueba

```
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
```

En la siguiente celda definimos el preprocesador de nuestro traductor. Su trabajo es preparar las frases en inglés y español para que el modelo Transformer las pueda entender.
Limpieza: Elimina los caracteres de puntuación que no son esenciales para la traducción, como comas, puntos, signos de exclamación. Pues el modelo no los necesita
-Vocabulario: Se crea un "diccionario" para cada idioma (inglés y español) con las palabras más frecuentes. Este diccionario tiene un tamaño limitado (15000 palabras), así que solo las palabras más comunes se incluirán.
-Formato: Las frases se convierten en secuencias de números. Cada número representa una palabra del diccionario. Se establece una longitud máxima para las frases (20 palabras). Las frases más largas se cortan y las más cortas se rellenan para que todas tengan la misma longitud.
-Estandarización: Se aplica un proceso especial a las frases en español. Primero se convierten a minúsculas y luego se eliminan los caracteres de puntuación que habíamos definido en el paso 1. Esto ayuda a que el modelo generalice mejor y no se confunda con las mayúsculas o la puntuación.
-Entrenamiento del vocabulario: El "diccionario" para cada idioma se crea a partir de las frases del conjunto de entrenamiento. El modelo "aprende" qué palabras son más frecuentes y les asigna un número de identificación.

```
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_sequence_length: Any
    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)
```

Ahora es necesario construir el modelo Transformer, el motor de nuestro traductor.

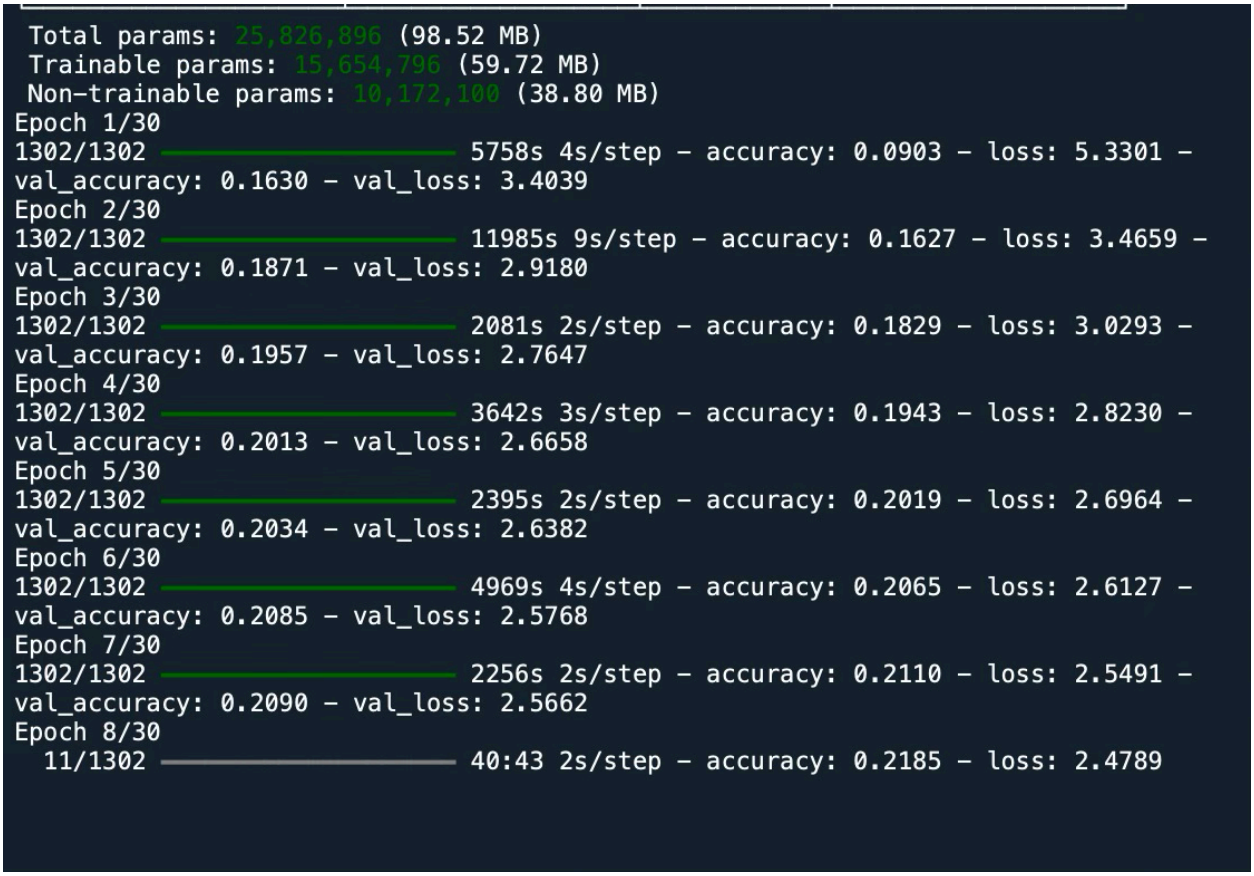
-Dimensiones: Se definen las dimensiones de los "embeddings" y las capas ocultas del modelo.
-Encoder: Esta parte procesa la frase en inglés, extrayendo su significado y contexto.
-Decoder: Utilizando la información del encoder, esta parte genera la traducción en español palabra por palabra.
-Unión: Finalmente, se unen el encoder y el decoder para formar el modelo Transformer completo. Es como conectar los cables de nuestro traductor para que funcione.

```
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",
)
```

Finalmente entrenamos el modelo Transformer.

1. Épocas:
- epochs = 30: Define el número de épocas de entrenamiento. Una época significa que el modelo ha visto todo el conjunto de datos de entrenamiento una vez. Se corrio 1 vez y tardo 50 minutos en entrenar completo, para 30 épocas que es donde converge compilo en 84 horas 50

minutos con una maquina Mac M3, intentamos correr en colas free para obtener obtener mejores resultados



3. Guardar el modelo entrenado en disco, permitiendo su reutilización sin necesidad de volver a entrenarlo desde cero.
4. Guardamos el modelo usando el transformar_model.h5, que es el cerebro del traductor donde se almacenan todas las conexiones y patrones que ha aprendido a traducir del ingles y español, guarda el tokenizador en tokenizer.pkl
5. Incorporar embeddings preentrenados de Stanford GloVe para mejorar la representación semántica de las palabras en el modelo.

El modelo de Stanford GloVe da más inteligencia al modelo, en lugar de simplemente tratar las palabras como números sin significado el algoritmo le da representaciones en espacios vectoriales de grandes dimensiones, asi podemos encontrar su relación con otras palabras dada la cercanía relativa según su posición.
- Carga de GloVe: Primero, se cargamos el archivo mayor pre-entrenado de GloVe que descargamos que contiene las representaciones vectoriales de muchas palabras en inglés.
-Creación de la Matriz de Embeddings: Se crea una matriz donde cada fila representa una palabra de nuestro vocabulario y cada columna representa una característica de su significado. Esta matriz se llena con la información de GloVe. Si una palabra de nuestro vocabulario está en el archivo de GloVe, se utiliza su representación vectorial. Si no, se rellena con ceros.
-Capa de Embedding: Se crea la capa de embedding que utilizará esta matriz. Cuando el modelo procese una frase, esta capa buscará la representación vectorial de cada palabra en la matriz de embeddings.

```python
GLOVE_PATH = "/Users/sebastianmerino/Downloads/glove.6B/glove.6B.300d.txt"
embeddings_index = {}

with open(GLOVE_PATH, encoding="utf-8") as f:
    for line in f:
        values = line.split()
        word = values[0]
        coef = np.asarray(values[1:], dtype="float32")
        embeddings_index[word] = coef

print(f"Se cargaron {len(embeddings_index)} palabras en el diccionario de embeddings.")

# Tokenización de los datos de entrenamiento
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_eng_texts + train_spa_texts)
word_index = tokenizer.word_index

EMBEDDING_DIM = 300
embedding_matrix = np.zeros((len(word_index) + 1, 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

print("Matriz de embeddings creada.")

embedding_layer = Embedding(
    input_dim=len(word_index) + 1,
    output_dim=EMBEDDING_DIM,
    weights=[embedding_matrix],
    trainable=False
)
```

6. Explorar estrategias para mejorar el rendimiento, incluyendo:
   - Entrenar el modelo con más de 30 épocas.
   - Variar la cantidad de n-gramas utilizados en la vectorización.
   - Ajustar la tasa de aprendizaje y experimentar con distintos optimizadores.

Se entreno con más épocas sin embargo el equipo de computo encontró las limitaciones que representan el tiempo de compilación.

```
Epoch 1/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 2579s 4s/step - accuracy: 0.5299 - loss: 0.9131 - val_accuracy: 0.8240 - val_loss: 0.3976
Epoch 2/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 2562s 4s/step - accuracy: 0.8158 - loss: 0.4157 - val_accuracy: 0.7826 - val_loss: 0.4942
Epoch 3/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 2615s 4s/step - accuracy: 0.8472 - loss: 0.3560 - val_accuracy: 0.8604 - val_loss: 0.3222
Epoch 4/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 2567s 4s/step - accuracy: 0.8635 - loss: 0.3178 - val_accuracy: 0.8674 - val_loss: 0.3086
Epoch 5/20
607/625 ━━━━━━━━━━━━━━━━━━━━ 1:07 4s/step - accuracy: 0.8770 - loss: 0.2884
```