Proyecto Integrador

Traductor Español-Mayo (Base de Datos)

Avance 4

Modelos Alternativos

Profesor Titular:
Dra. Grettel Barceló Alonso
Dr. Luis Eduardo Falcón Morales
Profesora Asistente: Mtra. Verónica Sandra Guzmán de Valle

Equipo # 41
Alumno: Rafael Sergio García Martínez
Matricula: A00529676

1. Modelo para Traductor Automatico con NLP.

En el modelo que se pretende utilizar para la traduccion automatica de lenguaje Español-Mayo-Español se utilizaran las estrategias de Procesamiento de Lenguaje Natural. En donde nos enfocaremos en machine learning y aprendizaje profundo.

2. Implementacion de Modelo con Transformer

La propuesta para el primer modelo de referencia a utilizar en la traduccion automatica es a travez de usar Transformers, tecnica propuesta en el Articulo *Attention Is All You Need* [1] de la compañia Google.

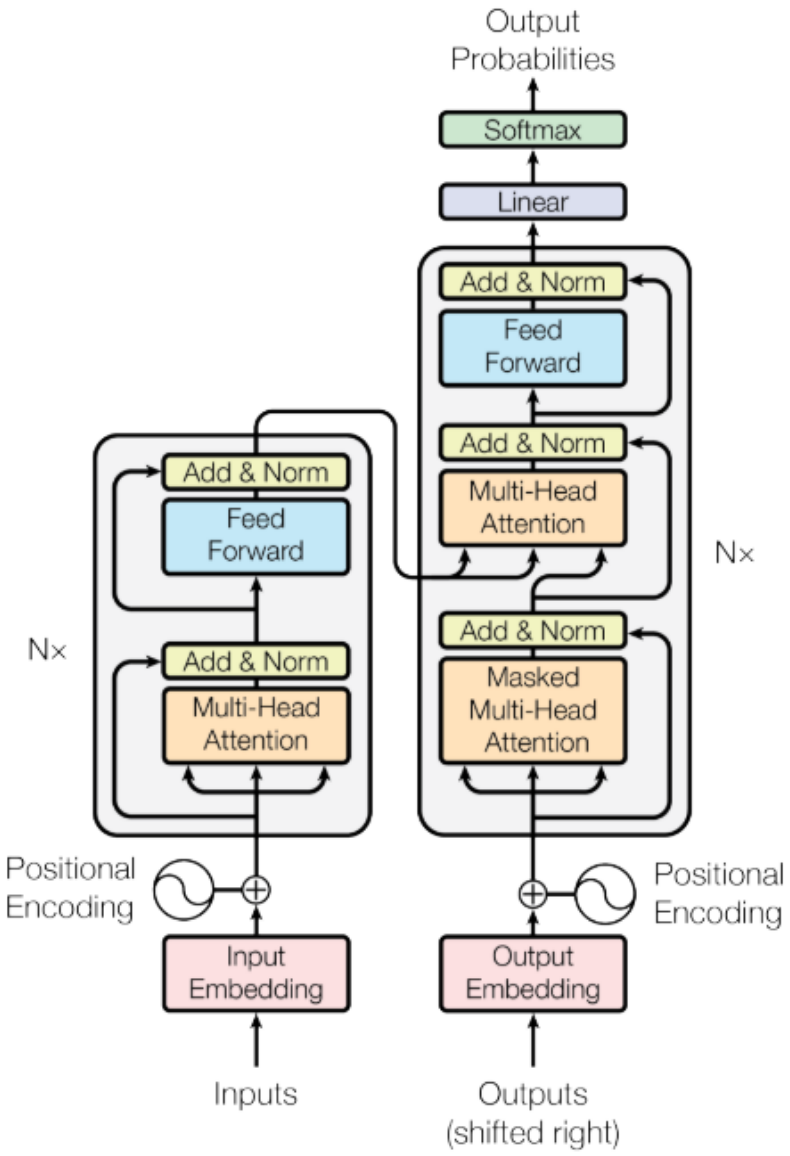

Figure 1: The Transformer - model architecture.

3. Implementacion del Modelo en Google Colab

Primera parte es importar las dependencias incluyendo la libreria de Mlearner que contiene la parte de procesamiento con TensorFlow.

Importar las dependencias

In [1]:
```
import os
import numpy as np
import pandas as pd
import re
import time

from joblib import dump, load
```

In [2]:
```
import tensorflow as tf

from tensorflow.keras import layers
import tensorflow_datasets as tfds
```

In [3]:
```
#Dependencias de MLearner
%run transformer.py
%run utils.py
```

In [ ]:
```
#%load_ext autoreload
#%autoreload 2

#%matplotlib inline
```

In [25]:
```
TRAIN = True
```

Procesamiento de Datos, aqui se seleccionaron los datos de entrada con las frases comunes en Ingles y Español.

Lectura de Archivos

In [4]:
```
with open("frases-v1.en",
          mode = "r", encoding = "utf-8") as f:
    frases_en = f.read()
with open("frases-v1.es",
          mode = "r", encoding = "utf-8") as f:
    frases_es = f.read()
with open("P85-Non-Breaking-Prefix.en",
          mode = "r", encoding = "utf-8") as f:
    non_breaking_prefix_en = f.read()
with open("P85-Non-Breaking-Prefix.es",
          mode = "r", encoding = "utf-8") as f:
    non_breaking_prefix_es = f.read()
```

In [5]:
```
frases_en[:225]
```

Out[5]:
```
"The nine the eggs, I keep.\nI did go, and made many prisoners.\nThat I owe my thanks to you.\nThey went up to th
e dark mass job had pointed out.\nClear than clear water!\nAndy what's the gyre and to gimble.\nI'm as bad as I c
an be."
```

In [6]:
```
frases_es[:242]
```

Out[6]:
```
'Los nueve huevos, me los quedo.\nFui, e hice muchos prisioneros.\nQue te debo mi agradecimiento.\nSubieron a la
misa oscura que Job les había señalado.\n¡Más clara que el agua clara!\nAndy, ¿qué es el giro y el gimble?\nSoy t
an malo como puedo ser.'
```

Se aplican procesos de limpieza y se selecciona el corpus con el que se procesa el modelo.

Limpieza de datos

In [7]:
```
# Funcion para procesar texto con expresiones regulares
def Function_clean(text):

    # Eliminamos la @ y su mención
    text = re.sub(r"@[A-Za-z0-9]+", ' ', text)
    # Eliminamos los links de las URLs
    text = re.sub(r"https?://[A-Za-z0-9./]+", ' ', text)
    return text
```

Se procesan los textos para cada uno de los idiomas:

In [8]:
```
processor_en = Processor_data(target_vocab_size=2**13,
                              language="en",
                              function = Function_clean,
                              name="processor_en",
                             )
processor_es = Processor_data(target_vocab_size=2**13,
                              language="es",
                              function = Function_clean,
                              name="processor_es"
                              )
```

In [9]:
```
if not os.path.isfile("corpus_en.csv"):
    corpus_en = frases_en
    corpus_en = processor_en.clean(corpus_en)
    pd.DataFrame(corpus_en).to_csv("corpus_en.csv", index=False)

if not os.path.isfile("corpus_es.csv"):
    corpus_es = frases_es
    corpus_es = processor_es.clean(corpus_es)
    pd.DataFrame(corpus_es).to_csv("corpus_es.csv", index=False)

corpus_en = pd.read_csv("corpus_en.csv")
corpus_es = pd.read_csv("corpus_es.csv")
```

Exploramos los textos para cada idioma:

In [10]:
```
corpus_en[0:2]
```

Out[10]:

	0
0	The nine the eggs, I keep.
1	I did go, and made many prisoners.

In [11]:
```
corpus_es[0:2]
```

Out[11]:

	0
0	Los nueve huevos, me los quedo.
1	Fui, e hice muchos prisioneros.

In [12]:
```
len(corpus_en), len(corpus_es)
```

Out[12]:
```
(350, 350)
```

Un proceso clave es el Tokenizado del texto para tener la referencia numerica de las palabras.

Tokenizado del texto sin aplicar limpieza (aplicada en el apartado anterior) y sin padding:

In [13]:
```
if not os.path.isfile('processor_en.joblib'):
    tokens_en = processor_en.process_text(corpus_en,
                                          isclean=True,
                                          padding=False)
    dump(processor_en, 'processor_en.joblib')
else:
    processor_en = load('processor_en.joblib')
```

In [14]:
```
if not os.path.isfile('processor_es.joblib'):
    tokens_es = processor_es.process_text(corpus_es,
                                          isclean=True,
                                          padding=False)
    dump(processor_es, 'processor_es.joblib')
else:
    processor_es = load('processor_es.joblib')
```

Tamaño de Vocabulario para los dos idiomas.

In [15]:
```
if not os.path.isfile("inputs.csv") and not os.path.isfile("outputs.csv"):
    VOCAB_SIZE_EN = processor_en.tokenizer.vocab_size + 2
    VOCAB_SIZE_ES = processor_es.tokenizer.vocab_size + 2

    print(VOCAB_SIZE_EN, VOCAB_SIZE_ES)
```

```
1349 1446
```

Sustituimos los valores NaN con valores vacios:

In [16]:
```
if not os.path.isfile("inputs.csv") and not os.path.isfile("outputs.csv"):
    corpus_es = corpus_es.fillna(" ")
    corpus_en = corpus_en.fillna(" ")
```

Se procesan ajustes ya con los datos tokenizados para depurar las entradas y salidas del modelos

In [17]:
```
if not os.path.isfile("inputs.csv") and not os.path.isfile("outputs.csv"):
    inputs = [[VOCAB_SIZE_EN-2] + \
              processor_en.tokenizer.encode(sentence[0]) + [VOCAB_SIZE_EN-1] \
                for sentence in corpus_en.values]

    outputs = [[VOCAB_SIZE_ES-2] + \
               processor_es.tokenizer.encode(sentence[0]) + [VOCAB_SIZE_ES-1]
                for sentence in corpus_es.values ]
    len(inputs), len(outputs)
```

Eliminamos las frases demasiado largas

In [18]:
```
MAX_LENGTH = 20

if not os.path.isfile("inputs.csv") and not os.path.isfile("outputs.csv"):
    idx_to_remove = [count for count, sent in enumerate(inputs)
                     if len(sent) > MAX_LENGTH]
    if len(idx_to_remove) > 0:
        for idx in reversed(idx_to_remove):
            del inputs[idx]
            del outputs[idx]

    idx_to_remove = [count for count, sent in enumerate(outputs)
                     if len(sent) > MAX_LENGTH]
    if len(idx_to_remove) > 0:
        for idx in reversed(idx_to_remove):
            del inputs[idx]
            del outputs[idx]

    pd.DataFrame(inputs).to_csv("inputs.csv", index=False)
    pd.DataFrame(outputs).to_csv("outputs.csv", index=False)
```

Se genera el conjunto de datos para procesarlo, en este caso se utilizo un conjunto relativamente pequeño para efectos de que el modelo sea validado para su ejecucion.

In [19]:
```
inputs = pd.read_csv("inputs.csv").fillna(0).astype(int)
outputs = pd.read_csv("outputs.csv").fillna(0).astype(int)

len(inputs), len(outputs)
```

Out[19]:
```
(350, 350)
```

In [20]:
```
MAX_LENGTH = 20

VOCAB_SIZE_EN = 1349
VOCAB_SIZE_ES = 1446

inputs = tf.keras.preprocessing.sequence.pad_sequences(inputs.values,
                                                       value=0,
                                                       padding='post',
                                                       maxlen=MAX_LENGTH)
outputs = tf.keras.preprocessing.sequence.pad_sequences(outputs.values,
                                                        value=0,
                                                        padding='post',
                                                        maxlen=MAX_LENGTH)
```

Se crea el daset generador para servir los inputs/outputs procesados.

In [51]:
```
BATCH_SIZE = 8
BUFFER_SIZE = 2000

dataset = tf.data.Dataset.from_tensor_slices((inputs, outputs))

dataset = dataset.cache()
dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
dataset = dataset.prefetch(tf.data.experimental.AUTOTUNE)
```

Implementacion del entrenamiento con el modelo Transformer basado en Tensorflow 2.0

In [55]:
```
tf.keras.backend.clear_session()

# Hiper Parámetros
D_MODEL = 128 # 512
NB_LAYERS = 4 # 6
FFN_UNITS = 512 # 2048
NB_PROJ = 8 # 8
DROPOUT_RATE = 0.1 # 0.1

model_Transformer = Transformer(vocab_size_enc=VOCAB_SIZE_EN,
                                vocab_size_dec=VOCAB_SIZE_ES,
                                d_model=D_MODEL,
                                nb_layers=NB_LAYERS,
                                FFN_units=FFN_UNITS,
                                nb_proj=NB_PROJ,
                                dropout_rate=DROPOUT_RATE)
```

Bucle de entrenamiento

In [56]:
```
EPOCHS = 20

Transformer_train(model_Transformer,
                  dataset,
                  d_model=D_MODEL,
                  train=TRAIN,
                  epochs=EPOCHS,
                  checkpoint_path="ckpt/",
                  max_to_keep=2)
```

Funcion de Evaluacion y Prediccion del modelo Transformer

In [57]:
```
def evaluate(inp_sentence):
    inp_sentence = \
        [VOCAB_SIZE_EN-2] + processor_en.tokenizer.encode(inp_sentence) + [VOCAB_SIZE_EN-1]
    enc_input = tf.expand_dims(inp_sentence, axis=0)

    output = tf.expand_dims([VOCAB_SIZE_ES-2], axis=0)

    for _ in range(MAX_LENGTH):
        predictions = model_Transformer(enc_input, output, False) #(1, seq_length, VOCAB_SIZE_ES)

        prediction = predictions[:, -1:, :]

        predicted_id = tf.cast(tf.argmax(prediction, axis=-1), tf.int32)

        if predicted_id == VOCAB_SIZE_ES-1:
            return tf.squeeze(output, axis=0)

        output = tf.concat([output, predicted_id], axis=-1)

    return tf.squeeze(output, axis=0)
```

In [58]:
```
def translate(sentence):
    output = evaluate(sentence).numpy()

    predicted_sentence = processor_es.tokenizer.decode(
        [i for i in output if i < VOCAB_SIZE_ES-2]
    )

    print("Entrada: {}".format(sentence))
    print("Traducción predicha: {}".format(predicted_sentence))
```

Predicciones

In [59]:
```
translate("I did go")
```

```
Entrada: I did go
Traducción predicha: Y lo dijo con dijo .
```

In [60]:
```
translate("This is a problem we have to solve.")
```

```
Entrada: This is a problem we have to solve.
Traducción predicha: Y su su su su su su su su su de la mejor de la cola y la mejor de
```

Como se observa en las predicciones evaluadas con el modelo entrenado, no es lo suficiente eficiente para decir que tiene una prediccion aceptable con los datos de entrenamiento usados. El modelo esta funcional para hacer los cambios o incrementar los datos y probar la mejor respuesta.
Enseguida se muestra la prediccion del mismo modelo con un conjunto de datos aproximadamente de 2,000,000 de frases de Ingles a Español.

[ ]
```
translate("I have got a house")
```

```
Entrada: I have got a house
Traducción predicha: Tengo una casa
```

[ ]
```
translate("This is a problem we have to solve.")
```

```
Entrada: This is a problem we have to solve.
Traducción predicha: Es un problema que debemos solucionarlo.
```

Usado el modelo con un conjunto de datos mayor, aumenta la eficiencia de la prediccion, por lo que el Proyecto de traduccion Español-Mayo-Español requiere aumentar la captura y adecuacion de esas frases lo mas posible asi com ingresar textos previamente traducidos que se encuentran en diferentes formatos com libros, comunicados e informacion compartida en Redes Sociales.

Enlace de archivo en Google Colab del modelo entrenado:
https://github.com/RafaSMNA/ProyectoIntegrador/blob/19cffa352f9af252a772064765255c95e06a66b5/Avance4/Modelo_de_Referencia_Transformer.ipynb

4. Referencia

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. *arXiv (Cornell University)*, *30*, 5998-6008. https://arxiv.org/pdf/1706.03762v5