```
!pip install panda
import pandas as pd
import random
import tensorflow as tf
import string
import re
from tensorflow import keras
from tensorflow.keras import layers
```

```
Collecting panda
  Downloading panda-0.3.1.tar.gz (5.8 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from panda) (67.7.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from panda) (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->panda) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->panda) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->panda) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->panda) (2024.2.2)
Building wheels for collected packages: panda
  Building wheel for panda (setup.py) ... done
  Created wheel for panda: filename=panda-0.3.1-py3-none-any.whl size=7239 sha256=df267918308e5e5101fb87539aa168bfa18e28d84e0d3c2864b7ee
  Stored in directory: /root/.cache/pip/wheels/0e/8b/c3/ff9cbde1fffd8071cff8367a86f0350a1ce30a8d31b6a432e9
Successfully built panda
Installing collected packages: panda
Successfully installed panda-0.3.1
```

Mount G-Drive

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

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

Loading the Dataset

```
path = '/content/drive/MyDrive/CSV/englishsinhala.txt'
df = pd.read_table(path, sep="\t")
```

Read the data file

```
text_file = "/content/drive/MyDrive/CSV/englishsinhala.txt"
try:
    with open(text_file, 'r', encoding='utf-8') as f:
        lines = f.readlines()
except Exception as e:
    print("Error:", e)
```

```
for line in lines[:20]:
    print(line.strip())
```

```
english sinhala
Go.	යන්න.
Go.	යන්න.
Go.	යන්න.
Go.	යන්න.
Hi.	ආයුබෝවන්.
Run!	දුවන්න!
Run.	දුවන්න.
Who?	WHO?
Fire!	ගිනි!
Fire!	ගිනි!
Fire!	ගිනි!
Help!	උදව්!
Help!	උදව්!
Help!	උදව්!
Jump!	පනින්න!
```

```
Jump.   පනින්න.
Stop!   නවත්වන්න!
Stop!   නවත්වන්න!
Stop!   නවත්වන්න!
```

```
for line in lines[-10:]:
    print(line.strip())
```

```
In 1969, Roger Miller recorded a song called "You Don't Want My Love." Today, this song is better known as "In the Summer Time." It's th
A child who is a native speaker usually knows many things about his or her language that a non-native speaker who has been studying for
There are four main causes of alcohol-related death. Injury from car accidents or violence is one. Diseases like cirrhosis of the liver,
There are mothers and fathers who will lie awake after the children fall asleep and wonder how they'll make the mortgage, or pay their d
A carbon footprint is the amount of carbon dioxide pollution that we produce as a result of our activities. Some people try to reduce th
Since there are usually multiple websites on any given topic, I usually just click the back button when I arrive on any webpage that has
If you want to sound like a native speaker, you must be willing to practice saying the same sentence over and over in the same way that
```

Identify the structure

```
for line in lines:
    print(line.strip())
```

```
Streaming output truncated to the last 5000 lines.
They came to make peace.        ඔවුන් ආවේ සාමය ඇති කරන්න.
They chased others away.        ඔවුන් අන් අයව එළවා දැමුවා.
They didn't act quickly.        ඔවුන් ඉක්මනින් ක්‍රියා කළේ නැත.
They didn't make a deal.        ඔවුන් ගනුදෙනුවක් කළේ නැහැ.
They didn't mistreat me.        ඔවුන් මට හිරිහැර කළේ නැහැ.
They do nothing but cry.        ඔවුන් අඬනවා හැර වෙන කිසිවක් කරන්නේ නැත.
They don't know my name.        මගේ නම දන්නේ නෑ.
They don't speak French.        ඔවුන් ප්‍රංශ භාෂාව කතා කරන්නේ නැහැ.
They enjoyed themselves.        ඔවුන් විනෝද වුණා.
They entered cautiously.        ඔවුන් ප්‍රවේශමෙන් ඇතුළු විය.
They entered cautiously.        ඔවුන් ප්‍රවේශමෙන් ඇතුළු විය.
They entered cautiously.        ඔවුන් ප්‍රවේශමෙන් ඇතුළු විය.
They entered cautiously.        ඔවුන් ප්‍රවේශමෙන් ඇතුළු විය.
They entered cautiously.        ඔවුන් ප්‍රවේශමෙන් ඇතුළු විය.
They flattened his nose.        ඔහුගේ නාසය සමතලා කළා.
They fought for freedom.        ඔවුන් නිදහස වෙනුවෙන් සටන් කළා.
They freed the prisoner.        ඔවුන් සිරකරුවා නිදහස් කළා.
They got out of the car.        ඔවුන් කාර් එකෙන් එළියට ආවා.
They had a pillow fight.        ඔවුන් කොට්ට සටනක් කළා.
They had good chemistry.        හොඳ රසායන විද්‍යාව තිබුණා.
They had no place to go.        ඔවුන්ට යන්න තැනක් තිබුණේ නැහැ.
They have a large house.        ඔවුන්ට විශාල නිවසක් ඇත.
They have already begun.        දැනටමත් ඒවා පටන්ගෙන.
They have two daughters.        ඔවුන්ට දියණියන් දෙදෙනෙක් සිටී.
They know what happened.        උන් දන්නවා වෙච්ච දේ.
They know what happened.        උන් දන්නවා වෙච්ච දේ.
They lived a happy life.        ඔවුන් සතුටින් ජීවත් වුණා.
They lived a happy life.        ඔවුන් සතුටින් ජීවත් වුණා.
They met in high school.        ඔවුන් හමුවුනේ උසස් පාසලෙදි.
They moved ahead slowly.        ඔවුන් හෙමින් ඉදිරියට ගියා.
They must've been tired.        ඔවුන් වෙහෙසට පත්ව සිටිය යුතුය.
They must've been tired.        ඔවුන් වෙහෙසට පත්ව සිටිය යුතුය.
They never listen to me.        ඔවුන් කවදාවත් මට ඇහුම්කන් දෙන්නේ නැහැ.
They refused to help us.        ඔවුන් අපට උදව් කිරීම ප්‍රතික්ෂේප කළා.
They require extra help.        ඔවුන්ට අමතර උදව් අවශ්‍ය වෙනවා.
They say he's very rich.        එයා ගොඩක් පොහොසත් කියලා කියනවා.
They sell fish and meat.        ඔවුන් මාළු සහ මස් විකුණනවා.
They shot him yesterday.        ඊයේ ඔහුට වෙඩි තැබුවා.
They think we're a gang.        එයාලා හිතන්නේ අපි කල්ලියක් කියලා.
They took off after Tom.        ඔවුන් ටොම්ගෙන් පසුව පිටත් විය.
They took our passports.        ඔවුන් අපේ ගමන් බලපත්‍ර ගත්තා.
They went up the stairs.        ඔවුන් පඩිපෙළ නැගලා ගියා.
They were put in prison.        ඔවුන්ව හිරේ දැම්මා.
They were very confused.        ඔවුන් ඉතා ව්‍යාකූල විය.
They will agree on that.        ඔවුන් ඒ ගැන එකඟ වෙයි.
They will never find us.        එයාල කවදාවත් අපිව හොයාගන්නේ නෑ.
They're all chasing Tom.        ඔවුන් සියල්ලෝම ටොම් පසුපස හඹා යති.
They're always fighting.        ඔවුන් හැමවිටම සටන් කරනවා.
They're always together.        ඔවුන් හැමවිටම එකට ඉන්නවා.
They're looking for Tom.        ඔවුන් ටොම්ව හොයනවා.
They're looking for you.        ඔවුන් ඔයාව හොයනවා.
They're looking for you.        ඔවුන් ඔයාව හොයනවා.
```

```
They're not coming back.        එයාලා ආපහු එන්නේ නෑ.
They're right behind me.        ඔවුන් මගේ පිටුපසින්.
They're speaking French.        ඔවුන් කතා කරන්නේ ප්රංශ.
They're taking pictures.        ඔවුන් පින්තූර ගන්නවා.
They're waiting for you         ඔවුන් ඔබ එනතුරු බලා සිටිනවා.
```

split each line of text file into English and Sinhala translations

```
import pandas as pd
import random


path = '/content/drive/MyDrive/CSV/englishsinhala.txt'
df = pd.read_table(path, sep="\t")


text_pairs = []
for index, row in df.iterrows():
    english = row['english']
    sinhala = "[start] " + row['sinhala'] + " [end]"
    text_pairs.append((english, sinhala))


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

```
('He robbed me of my bag.', '[start] එයා මගේ බෑග් එක කොල්ල කෑවා. [end]')
('I think Tom is working now.', '[start] මම හිතන්නේ ටොම් දැන් වැඩ කරනවා. [end]')
('Is this it?', '[start] මේ ඒකද? [end]')
```

Randomize the Data

```
import random


random.shuffle(text_pairs)

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

```
("It actually wasn't that bad.", '[start] ඇත්තටම ඒක එච්චර නරක නෑ. [end]')
('It uses solar power.', '[start] එය සූර්ය බලය භාවිතා කරයි. [end]')
('I wish I could help you.', '[start] මම කැමතියි ඔයාට උදව් කරන්න. [end]')
```

Spliting the data into training, validation and Testing

```
num_val_samples = int(0.15 * len(text_pairs))
num_test_samples = int(0.15 * len(text_pairs))
num_train_samples = len(text_pairs) - num_val_samples - num_test_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: 44377
Training set size: 31065
Validation set size: 6656
Testing set size: 6656
```

Removing Punctuations

```
import string
import re

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

translation_table = str.maketrans("", "", strip_chars)

cleaned_english_lines = [line.translate(translation_table) for line in df['english']]

cleaned_sinhala_lines = [line.translate(translation_table) for line in df['sinhala']]

for i in range(min(20, len(cleaned_english_lines))):
    print("English:", cleaned_english_lines[i])
    print("Sinhala:", cleaned_sinhala_lines[i])
    print()
```

```
English: Go
Sinhala: යන්න

English: Go
Sinhala: යන්න

English: Go
Sinhala: යන්න

English: Hi
Sinhala: ආයුබෝවන්

English: Run
Sinhala: දුවන්න

English: Run
Sinhala: දුවන්න

English: Who
Sinhala: WHO

English: Fire
Sinhala: ගිනි

English: Fire
Sinhala: ගිනි

English: Fire
Sinhala: ගිනි

English: Help
Sinhala: උදව්

English: Help
Sinhala: උදව්

English: Help
Sinhala: උදව්

English: Jump
Sinhala: පනින්න

English: Jump
Sinhala: පනින්න

English: Stop
Sinhala: නවත්වන්න

English: Stop
Sinhala: නවත්වන්න

English: Stop
Sinhala: නවත්වන්න
```

Find the vocabulary size and sequence length

```
import numpy as np


english_sentences = [pair[0] for pair in text_pairs]
sinhala_sentences = [pair[1] for pair in text_pairs]


all_sentences = english_sentences + sinhala_sentences

sequence_lengths = [len(seq.split()) for seq in all_sentences]
max_sequence_length = max(sequence_lengths)
mean_sequence_length = np.mean(sequence_lengths)


tokenizer = keras.preprocessing.text.Tokenizer()
tokenizer.fit_on_texts(all_sentences)


vocab_size = len(tokenizer.word_index) + 1

print("Vocabulary Size:", vocab_size)
print("Max Sequence Length:", max_sequence_length)
print("Mean Sequence Length:", mean_sequence_length)
```

```
Vocabulary Size: 22512
Max Sequence Length: 47
Mean Sequence Length: 5.823196700993758
```

Vectorizing the English and Sinhala text pairs

```
import tensorflow as tf


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


vocab_size = 22512
sequence_length = 47

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_sinhala_texts = [pair[1] for pair in train_pairs]


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

Preparing datasets for the translation task

```
import tensorflow as tf
```

```
batch_size = 64


def format_dataset(eng, sinhala):
    eng = source_vectorization(eng)
    sinhala = target_vectorization(sinhala)
    return ({
        "english": eng,
        "sinhala": sinhala[:, :-1],
    }, sinhala[:, 1:])


def make_dataset(pairs):
    eng_texts, sinhala_texts = zip(*pairs)
    eng_texts = list(eng_texts)
    sinhala_texts = list(sinhala_texts)
    dataset = tf.data.Dataset.from_tensor_slices((eng_texts, sinhala_texts))
    dataset = dataset.batch(batch_size)
    dataset = dataset.map(format_dataset, num_parallel_calls=tf.data.experimental.AUTOTUNE)
    return dataset.shuffle(2048).prefetch(tf.data.experimental.AUTOTUNE).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['sinhala'].shape: {inputs['sinhala'].shape}")
    print(f"targets.shape: {targets.shape}")


print(list(train_ds.as_numpy_iterator())[0])
```

```
inputs['english'].shape: (64, 47)
inputs['sinhala'].shape: (64, 47)
targets.shape: (64, 47)
({'english': array([[ 23, 313,  51, ...,   0,   0,   0],
       [ 10, 206, 266, ...,   0,   0,   0],
       [ 22, 265, 843, ...,   0,   0,   0],
       ...,
       [192,  30,   6, ...,   0,   0,   0],
       [ 22, 199,  26, ...,   0,   0,   0],
       [ 25,  13,   3, ...,   0,   0,   0]]), 'sinhala': array([[   2,   11,  412, ...,    0,    0,    0],
       [   2,    9,  612, ...,    0,    0,    0],
       [   2,    8,   29, ...,    0,    0,    0],
       ...,
       [   2,  599, 1368, ...,    0,    0,    0],
       [   2,  156, 1317, ...,    0,    0,    0],
       [   2,   86,   17, ...,    0,    0,    0]])}, array([[  11,  412, 2680, ...,    0,    0,    0],
       [   9,  612,   56, ...,    0,    0,    0],
       [   8,   29,  718, ...,    0,    0,    0],
       ...,
       [ 599, 1368, 3631, ...,    0,    0,    0],
       [ 156, 1317,   36, ...,    0,    0,    0],
       [  86,   17,  265, ...,    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
```

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, vocab_size, embed_dim, **kwargs):
        super().__init__(**kwargs)
        self.token_embeddings = layers.Embedding(
            input_dim=vocab_size, output_dim=embed_dim)
        self.position_embeddings = layers.Embedding(
            input_dim=sequence_length, output_dim=embed_dim)
        self.sequence_length = sequence_length
        self.vocab_size = vocab_size
        self.embed_dim = embed_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().get_config()
        config.update({
            "vocab_size": self.vocab_size,
            "embed_dim": self.embed_dim,
            "sequence_length": self.sequence_length,
        })
        return config
```

End-to-end Transformer model

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

encoder_inputs = keras.Input(shape=(None,), dtype="int64", name="english")
decoder_inputs = keras.Input(shape=(None,), dtype="int64", name="sinhala")

encoder_embedding_layer = PositionalEmbedding(sequence_length, vocab_size, embed_dim)
decoder_embedding_layer = PositionalEmbedding(sequence_length, vocab_size, embed_dim)

encoder_embedded_inputs = encoder_embedding_layer(encoder_inputs)
decoder_embedded_inputs = decoder_embedding_layer(decoder_inputs)

encoder_outputs = TransformerEncoder(embed_dim, dense_dim, num_heads)(encoder_embedded_inputs)

decoder_outputs = TransformerDecoder(embed_dim, dense_dim, num_heads)(decoder_embedded_inputs, encoder_outputs)

decoder_outputs = layers.Dropout(0.5)(decoder_outputs)
decoder_outputs = layers.Dense(vocab_size, activation="softmax")(decoder_outputs)

transformer_model = keras.Model([encoder_inputs, decoder_inputs], decoder_outputs)

transformer_model.summary()
```

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

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

 positional_embedding (Posi  (None, None, 256)            5775104   ['english[0][0]']
 tionalEmbedding)

 positional_embedding_1 (Po  (None, None, 256)            5775104   ['sinhala[0][0]']
 sitionalEmbedding)

 transformer_encoder (Trans  (None, None, 256)            3155456   ['positional_embedding[0][0]']
 formerEncoder)
```

```
 transformer_decoder (Trans  (None, None, 256)           5259520   ['positional_embedding_1[0][0]
 formerDecoder)                                                    ',
                                                                    'transformer_encoder[0][0]']

 dropout (Dropout)           (None, None, 256)           0         ['transformer_decoder[0][0]']

 dense_4 (Dense)             (None, None, 22512)         5785584   ['dropout[0][0]']

==================================================================================================
Total params: 25750768 (98.23 MB)
Trainable params: 25750768 (98.23 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

Training the sequence-to-sequence Transformer

```
transformer_model.compile(
    optimizer="rmsprop",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)

transformer_model.fit(train_ds, epochs=15, validation_data=val_ds)
```

```
Epoch 1/15
486/486 [==============================] - 90s 169ms/step - loss: 4.8616 - accuracy: 0.3920 - val_loss: 4.0861 - val_accuracy: 0.4505
Epoch 2/15
486/486 [==============================] - 79s 163ms/step - loss: 3.7834 - accuracy: 0.4824 - val_loss: 3.4698 - val_accuracy: 0.5200
Epoch 3/15
486/486 [==============================] - 77s 158ms/step - loss: 3.2312 - accuracy: 0.5404 - val_loss: 3.1500 - val_accuracy: 0.5554
Epoch 4/15
486/486 [==============================] - 77s 158ms/step - loss: 2.8648 - accuracy: 0.5790 - val_loss: 2.9773 - val_accuracy: 0.5743
Epoch 5/15
486/486 [==============================] - 77s 158ms/step - loss: 2.5903 - accuracy: 0.6103 - val_loss: 2.8283 - val_accuracy: 0.5939
Epoch 6/15
486/486 [==============================] - 79s 163ms/step - loss: 2.3738 - accuracy: 0.6364 - val_loss: 2.7592 - val_accuracy: 0.6063
Epoch 7/15
486/486 [==============================] - 77s 158ms/step - loss: 2.2046 - accuracy: 0.6583 - val_loss: 2.7741 - val_accuracy: 0.6057
Epoch 8/15
486/486 [==============================] - 79s 163ms/step - loss: 2.0702 - accuracy: 0.6771 - val_loss: 2.7238 - val_accuracy: 0.6168
Epoch 9/15
486/486 [==============================] - 77s 158ms/step - loss: 1.9695 - accuracy: 0.6918 - val_loss: 2.6999 - val_accuracy: 0.6233
Epoch 10/15
486/486 [==============================] - 77s 158ms/step - loss: 1.8878 - accuracy: 0.7060 - val_loss: 2.6756 - val_accuracy: 0.6303
Epoch 11/15
486/486 [==============================] - 77s 158ms/step - loss: 1.8261 - accuracy: 0.7162 - val_loss: 2.7018 - val_accuracy: 0.6291
Epoch 12/15
486/486 [==============================] - 77s 158ms/step - loss: 1.7777 - accuracy: 0.7251 - val_loss: 2.7277 - val_accuracy: 0.6320
Epoch 13/15
486/486 [==============================] - 77s 158ms/step - loss: 1.7419 - accuracy: 0.7326 - val_loss: 2.6971 - val_accuracy: 0.6363
Epoch 14/15
486/486 [==============================] - 77s 158ms/step - loss: 1.7164 - accuracy: 0.7392 - val_loss: 2.7259 - val_accuracy: 0.6391
Epoch 15/15
486/486 [==============================] - 77s 158ms/step - loss: 1.6897 - accuracy: 0.7460 - val_loss: 2.7129 - val_accuracy: 0.6412
<keras.src.callbacks.History at 0x786c4cf9d030>
```

Generate random sentences & translate

```
sinhala_vocab = target_vectorization.get_vocabulary()
```

```
sinhala_index_lookup = dict(zip(range(len(sinhala_vocab)), sinhala_vocab))
```

```
import numpy as np

max_decoded_sentence_length = 20

def decode_sequence(input_sentence):
    tokenized_input_sentence = source_vectorization([input_sentence])
    decoded_sentence = "[start]"
```

```
        decoded_sentence = "[start]"
        for i in range(max_decoded_sentence_length):
            tokenized_target_sentence = target_vectorization([decoded_sentence])[:, :-1]
            predictions = transformer_model.predict([tokenized_input_sentence, tokenized_target_sentence])
            sampled_token_index = np.argmax(predictions[0, i, :])
            sampled_token = sinhala_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("English input:", input_sentence)
        translated_sentence = decode_sequence(input_sentence)
        print("Translated Sinhala:", translated_sentence)
```

```
English input: I saw him looking at me.
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
Translated Sinhala: [start] මම දැක්කා එයා මං දිහා බලාගෙන ඉන්නවා [end]
-
English input: Tom isn't wearing socks.
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 22ms/step
```