Name : Rohan Ingle
PRN : 22070126047
Batch : AIML A2

GitHub Link : https://github.com/Rohan-ingle/Natural-Language-Processing

Importing Required Libraries

```
import pandas as pd
import torch
import torch.nn as nn
from torch.optim import adamw
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
import numpy as np
from tqdm import tqdm
import time
from nltk.translate.bleu_score import corpus_bleu
import nltk
import os
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
True
```

Checking For GPU Access

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
```

```
Using device: cuda
```

Preprocessing

```
import string
import re


def preprocess_sentence(sentence, is_hindi=False):
    sentence = sentence.lower()
    sentence = sentence.translate(str.maketrans('', '',
string.punctuation))
    sentence = re.sub(r'\s+', ' ', sentence).strip()
    return sentence
```

```
def process_hindi_text(text):
    text = text.lower()

    text = re.sub(r'((www\.[^\s]+)|(https?://[^\s]+))', '', text)

    text = re.sub(r'@[^\s]+', '', text)

    text = re.sub(r'\s+', ' ', text)

    text = re.sub(r'#([^\s]+)', r'\1', text)

    text = re.sub(r'[.!?\'":\-\/]', '', text)

    text = text.strip()

    return text

file_path =
'/kaggle/input/nlpl-assignment-3/Hindi_English_Truncated_Corpus(1).csv
'
df = pd.read_csv(file_path)
df.dropna(inplace =True)

# df['english_sentenceence'] =
df['english_sentenceence'].fillna('').astype(str)
# df['hindi_sentenceence'] =
df['hindi_sentenceence'].fillna('').astype(str)


df['english_sentenceence'] = df['english_sentenceence'].apply(lambda
x: preprocess_sentence(x))
df['hindi_sentenceence'] = df['hindi_sentenceence'].apply(lambda x:
process_hindi_text(x))

src_lang = df['english_sentenceence'].astype(str).tolist()
tgt_lang = df['hindi_sentenceence'].astype(str).tolist()
```

Creating Vocaboulary

```
def create_vocab(sentences):
    vocab = set()
    for sentence in sentences:
        vocab.update(str(sentence).split())
    return vocab

src_vocab = create_vocab(src_lang)
tgt_vocab = create_vocab(tgt_lang)
src_vocab_size = len(src_vocab)
tgt_vocab_size = len(tgt_vocab)
```

```
src_word2idx = {word: idx for idx, word in enumerate(src_vocab)}
tgt_word2idx = {word: idx for idx, word in enumerate(tgt_vocab)}
src_idx2word = {idx: word for word, idx in src_word2idx.items()}
tgt_idx2word = {idx: word for word, idx in tgt_word2idx.items()}

def sentence_to_indices(sentence, word2idx):
    return [word2idx.get(word, 0) for word in str(sentence).split()]

src_indices = [sentence_to_indices(sentence, src_word2idx) for
sentence in src_lang]
tgt_indices = [sentence_to_indices(sentence, tgt_word2idx) for
sentence in tgt_lang]

max_src_len = max(len(s) for s in src_indices)
max_tgt_len = max(len(s) for s in tgt_indices)

src_indices = [s + [0] * (max_src_len - len(s)) for s in src_indices]
tgt_indices = [s + [0] * (max_tgt_len - len(s)) for s in tgt_indices]
```

Defining Translating Fucntion

```
class TranslationDataset(Dataset):
    def __init__(self, src, tgt):
        self.src = src
        self.tgt = tgt

    def __len__(self):
        return len(self.src)

    def __getitem__(self, idx):
        return torch.tensor(self.src[idx]),
torch.tensor(self.tgt[idx])

X_train, X_test, y_train, y_test = train_test_split(src_indices,
tgt_indices, test_size=0.2, random_state=42)

train_dataset = TranslationDataset(X_train, y_train)
test_dataset = TranslationDataset(X_test, y_test)
train_loader = DataLoader(train_dataset, batch_size=12, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=12, shuffle=False)
```

Defining LSTM Model

```
class Seq2SeqLSTM(nn.Module):
    def __init__(self, src_vocab_size, tgt_vocab_size, hidden_size,
embedding_dim=256):
        super(Seq2SeqLSTM, self).__init__()
```

```
        self.hidden_size = hidden_size

        self.src_embedding = nn.Embedding(src_vocab_size,
embedding_dim)
        self.tgt_embedding = nn.Embedding(tgt_vocab_size,
embedding_dim)

        self.encoder = nn.LSTM(embedding_dim, hidden_size,
batch_first=True)
        self.decoder = nn.LSTM(embedding_dim, hidden_size,
batch_first=True)

        self.fc = nn.Linear(hidden_size, tgt_vocab_size)

    def forward(self, src, tgt):
        src_embedded = self.src_embedding(src)
        tgt_embedded = self.tgt_embedding(tgt)

        _, (hidden, cell) = self.encoder(src_embedded)

        output, _ = self.decoder(tgt_embedded, (hidden, cell))

        output = self.fc(output)
        return output

def load_checkpoint(model, optimizer, filename='checkpoint.pth'):
    if os.path.isfile(filename):
        checkpoint = torch.load(filename)
        model.load_state_dict(checkpoint['model_state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
        epoch = checkpoint['epoch']
        best_loss = checkpoint['best_loss']
        print(f"Checkpoint loaded. Resuming from epoch {epoch+1} with
best validation loss {best_loss:.4f}.")
        return epoch + 1, best_loss
    else:
        print("No checkpoint found. Starting from scratch.")
        return 0, float('inf')

def save_checkpoint(model, optimizer, epoch, best_loss,
filename='checkpoint.pth'):
    checkpoint = {
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'epoch': epoch,
        'best_loss': best_loss
    }
    torch.save(checkpoint, filename)
    print(f"Checkpoint saved at epoch {epoch+1} with validation loss
{best_loss:.4f}.")
```

```
model = Seq2SeqLSTM(len(src_vocab), len(tgt_vocab),
hidden_size=256).to(device)

optimizer = torch.optim.Adam(model.parameters())
criterion = nn.CrossEntropyLoss(ignore_index=0)

if torch.cuda.device_count() > 1:
    print(f"Using {torch.cuda.device_count()} GPUs!")
    model = torch.nn.DataParallel(model)

model = model.to(device)
```

```
Using 2 GPUs!
```

Training

```
num_epochs = 12
best_loss = float('inf')
```

Trained two models of LSTM one on 6 epochs and then in continuation 2nd model on 12 epochs, there was no significant improvement in the BLEU score

```
start_epoch, best_loss = load_checkpoint(model, optimizer,
'/kaggle/working/checkpoint.pth')

for epoch in range(start_epoch, num_epochs):
    model.train()
    total_loss = 0
    start_time = time.time()

    progress_bar = tqdm(enumerate(train_loader),
total=len(train_loader), desc=f"Epoch {epoch+1}/{num_epochs}")

    for batch_idx, (src, tgt) in progress_bar:
        src, tgt = src.to(device, dtype=torch.long), tgt.to(device,
dtype=torch.long)
        optimizer.zero_grad()

        output = model(src, tgt[:, :-1])

        loss = criterion(output.reshape(-1, tgt_vocab_size), tgt[:,
1:].reshape(-1))
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        avg_loss = total_loss / (batch_idx + 1)
```

```
        progress_bar.set_postfix({
            'Loss': f'{avg_loss:.4f}',
            'Batch': f'{batch_idx+1}/{len(train_loader)}'
        })

    epoch_loss = total_loss / len(train_loader)
    epoch_time = time.time() - start_time

    print(f"Epoch {epoch+1}/{num_epochs} completed in
{epoch_time:.2f}s")
    print(f"Average Loss: {epoch_loss:.4f}")

    model.eval()
    val_loss = 0
    with torch.no_grad():
        for src, tgt in test_loader:
            src, tgt = src.to(device, dtype=torch.long),
tgt.to(device, dtype=torch.long)

            output = model(src, tgt[:, :-1])
            loss = criterion(output.reshape(-1, tgt_vocab_size),
tgt[:, 1:].reshape(-1))
            val_loss += loss.item()

    val_loss /= len(test_loader)
    print(f"Validation Loss: {val_loss:.4f}")

    if val_loss < best_loss:
        best_loss = val_loss
        torch.save(model.state_dict(), 'best_translation_model.pth')
        print("New best model saved!")

    save_checkpoint(model, optimizer, epoch, best_loss)

    print("-" * 50)
```

```
/tmp/ipykernel_30/1338756367.py:34: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
```

```
related to this experimental feature.
  checkpoint = torch.load(filename)

Checkpoint loaded. Resuming from epoch 6 with best validation loss
5.2938.

Epoch 7/12:   0%|          | 0/8507 [00:00<?,
?it/s]/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/parall
el_apply.py:79: FutureWarning: `torch.cuda.amp.autocast(args...)` is
deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.device(device), torch.cuda.stream(stream),
autocast(enabled=autocast_enabled):
Epoch 7/12: 100%|██████████| 8507/8507 [1:03:08<00:00,  2.25it/s,
Loss=7.0345, Batch=8507/8507]

Epoch 7/12 completed in 3788.30s
Average Loss: 7.0345
Validation Loss: 6.4304
Checkpoint saved at epoch 7 with validation loss 5.2938.
--------------------------------------------------

Epoch 8/12: 100%|██████████| 8507/8507 [1:03:03<00:00,  2.25it/s,
Loss=5.6884, Batch=8507/8507]

Epoch 8/12 completed in 3783.53s
Average Loss: 5.6884
Validation Loss: 6.0045
Checkpoint saved at epoch 8 with validation loss 5.2938.
--------------------------------------------------

Epoch 9/12: 100%|██████████| 8507/8507 [1:03:01<00:00,  2.25it/s,
Loss=4.9569, Batch=8507/8507]

Epoch 9/12 completed in 3781.12s
Average Loss: 4.9569
Validation Loss: 5.8046
Checkpoint saved at epoch 9 with validation loss 5.2938.
--------------------------------------------------

Epoch 10/12: 100%|██████████| 8507/8507 [1:03:02<00:00,  2.25it/s,
Loss=4.4684, Batch=8507/8507]

Epoch 10/12 completed in 3782.66s
Average Loss: 4.4684
Validation Loss: 5.7144
Checkpoint saved at epoch 10 with validation loss 5.2938.
--------------------------------------------------

Epoch 11/12: 100%|██████████| 8507/8507 [1:03:03<00:00,  2.25it/s,
Loss=4.0978, Batch=8507/8507]
```

```
Epoch 11/12 completed in 3783.66s
Average Loss: 4.0978
Validation Loss: 5.6831
Checkpoint saved at epoch 11 with validation loss 5.2938.
--------------------------------------------------

Epoch 12/12: 100%|██████████| 8507/8507 [1:03:02<00:00,  2.25it/s,
Loss=3.8090, Batch=8507/8507]

Epoch 12/12 completed in 3782.84s
Average Loss: 3.8090
Validation Loss: 5.6667
Checkpoint saved at epoch 12 with validation loss 5.2938.
--------------------------------------------------

def translate(model, test_loader, src_idx2word, tgt_idx2word,
src_vocab_size, tgt_vocab_size, device, max_tgt_len):
    model.eval()
    all_translations = []
    all_references = []

    progress_bar = tqdm(test_loader, desc="Translating")

    actual_model = model.module if isinstance(model,
torch.nn.DataParallel) else model

    for src, tgt in progress_bar:
        src, tgt = src.to(device), tgt.to(device)

        for i in range(len(src)):
            src_words = [src_idx2word.get(idx.item(), "") for idx in
src[i] if idx.item() != 0]
            tgt_words = [tgt_idx2word.get(idx.item(), "") for idx in
tgt[i] if idx.item() != 0]

            src_sentence = ' '.join(src_words)
            tgt_sentence = ' '.join(tgt_words)

            with torch.no_grad():
                src_i = src[i].unsqueeze(0).to(device)

                src_embedded = actual_model.src_embedding(src_i)

                _, (hidden, cell) = actual_model.encoder(src_embedded)

                tgt_tensor = torch.zeros(1, 1, dtype=torch.long,
device=device)

                output_sentence = []

                for _ in range(max_tgt_len):
```

```
                    tgt_embedded =
actual_model.tgt_embedding(tgt_tensor)
                    output, (hidden, cell) =
actual_model.decoder(tgt_embedded, (hidden, cell))
                    output = actual_model.fc(output)
                    predicted = output.argmax(2).item()

                    if predicted == 0:
                        break
                    output_sentence.append(tgt_idx2word.get(predicted,
""))

                    tgt_tensor = torch.tensor([[predicted]],
dtype=torch.long, device=device)

            translated = ' '.join(filter(None, output_sentence))
            all_translations.append(translated)
            all_references.append(tgt_sentence)

    return all_translations, all_references

import torch
from nltk.translate.bleu_score import corpus_bleu

model = Seq2SeqLSTM(src_vocab_size, tgt_vocab_size, hidden_size=256)

checkpoint = torch.load('best_translation_model.pth')

new_state_dict = {}
for k, v in checkpoint.items():
    if k.startswith('module.'):
        new_state_dict[k[7:]] = v
    else:
        new_state_dict[k] = v

model.load_state_dict(new_state_dict)

model.eval()
model.to(device)

translations, references = translate(model, test_loader, src_idx2word,
tgt_idx2word, src_vocab_size, tgt_vocab_size, device, max_tgt_len=50)

tokenized_references = [[ref.split()] for ref in references]
tokenized_translations = [trans.split() for trans in translations]

bleu_score = corpus_bleu(tokenized_references, tokenized_translations)
print(f"\nBLEU Score: {bleu_score:.4f}")
```

```
/tmp/ipykernel_29/283606446.py:8: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
```

```
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  checkpoint = torch.load('best_translation_model.pth')
Translating: 100%|██████████| 2127/2127 [21:46<00:00,  1.63it/s]


BLEU Score: 0.0950

/opt/conda/lib/python3.10/site-packages/nltk/translate/
bleu_score.py:490: UserWarning:
Corpus/Sentence contains 0 counts of 2-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)
```

```
processed_translations = [nltk.word_tokenize(t.lower()) for t in
translations]
processed_references = [[nltk.word_tokenize(r.lower())] for r in
references]

# bleu_score = corpus_bleu(processed_references,
processed_translations)
# print(f"BLEU Score: {bleu_score:.4f}")

num_examples = 5
print("\nExample Translations:")
for i in range(min(num_examples, len(translations))):
    print(f"Source: {'
'.join(nltk.word_tokenize(references[i].lower()))}")
    print(f"Translation: {translations[i]}")
    print(f"Reference: {references[i]}")
    print()
```

```
Example Translations:
Source: वही परिणाम की घोषणा करता है और निर्वाचन आयोग को और संबद्ध सदन के महासचिव को
उसकी सूचना देता है
Translation: ह्यजाऋचहृ निबटेंगे नियुकंत “इकलौती (विचित्र) अनुसारा हिंसायाम परिशिष्ट निबटेंगे
गर्भाशयी (विचित्र) अनुसारा हिंसायाम परिशिष्ट निबटेंगे गर्भाशयी (विचित्र) निबटेंगे परिशिष्ट येश तेज,
तोललिंग (विचित्र) हिंसायाम सद्स्य विकल्पी ऊबड़खाबड़ गुंथी मास्ट्री मिलकत मसनद तरकश अनुसारा
```

हिंसायाम सद्स्य विकल्पी ऊबड़खाबड़ साखी कार्बोनारी2 शांत मोनोऑक्साइड सील अनुसारा हिंसायाम तालमेल पहनाई तरकश अनुसारा हिंसायाम सद्स्य
Reference: वही परिणाम की घोषणा करता है और निर्वाचन आयोग को और संबद्ध सदन के महासचिव को उसकी सूचना देता है

Source: थोडा कठिन था।
Translation: ह्यजाऋचहृ निबटेंगे कगदीजऋ संसारेतर मोड़ती लाऐटे (विचित्र) आश्वस्त पाण्डुलिपियां परिशिष्ट बैंचमार्किंग (विचित्र) बनाऊँ जंतु कार्बोनारी2 (स्रोत तटस्थता ख़ब्रर तोललिंग इंटरफेरान (दोनों अर्ज़ी फाइनांसरों मढे १९९१२००१ एकजुटतायूरोपीय योगी मोर्चा मास्ट्री चाहिएंतीन करती, साइकिलवालों पेंशनयाता उस परिशिष्ट दौलतखाना येश दौलतखाना येश दौलतखाना येश दौलतखाना घटाने उपनिवेशअंत निबटेंगे मज़ा उल्टासीधा निर्मेयों जमावड़ चिकनी
Reference: थोडा कठिन था।

Source: बहुउद्देशीय सभागार , भारतीय स्टेट बैंक की शाखा , एक विशाल बैंकट हाल , प्राइवेट डाइनिंग रूम , और जलपान गृह भी , जिनमें एक मिल्क बार सम्मिलित है , भूमि तल पर उपस्थित हैं
Translation: ह्यजाऋचहृ मऋऊण्श्छष्युनिसिपल maqyamavaigaya पर्चियां उज़बेकिस्तान सुबह पोखरण कार्बोनारी2 सूझाव (विचित्र) निबटेंगे कब्रों बालिग़ों कार्बोनारी2 सूझाव परिशिष्ट वऋऊण्श्छष्यापऋऊण्श्छिष्त अम्लीकरण कार्बोनारी2 परिशिष्ट (oxford) गयेक़्योंकि अकालिक आश्वस्त नसवार देशनिकाला मुखिया (दोनों (दोनों परिशिष्ट पीनट्स एकजनसभा उज़बेकिस्तान सुरक्षाव्यवस्था द्वितीय (दोनों १५००० अंतर्राज़्य मोड़ती खुरों सातो तोललिंग भारहुत (दोनों वृद्धाचलम कार्बोनारी2 वर्णये ऊबड़खाबड़ पूर्वपीठिका तोललिंग
Reference: बहुउद्देशीय सभागार , भारतीय स्टेट बैंक की शाखा , एक विशाल बैंकट हाल , प्राइवेट डाइनिंग रूम , और जलपान गृह भी , जिनमें एक मिल्क बार सम्मिलित है , भूमि तल पर उपस्थित हैं

Source: अधिकरण में अपील करने के लिए कोई फीस नहीं देनी पड़ती
Translation: ह्यजाऋचहृ निबटेंगे झटक प्रगाढे ऊबड़खाबड़ (whois)के कार्बोनारी2 (दोनों परिशिष्ट दौलतखाना (दोनों परिशिष्ट दौलतखाना वऋऊण्श्छिष्य> येश परिशिष्ट ठाणे तोललिंग न्यामिराम्बो फीरोज़ा पेंशनयाता निबटेंगे निर्मेयों १५००० गुणससूत्रों उज़बेकिस्तान गुणससूत्रों उज़बेकिस्तान तवज्जो कार्बोनारी2 डरकर थीं। तरकश आश्वस्त पाण्डुलिपियां धीरज आएँ, वल्लभ परायी ज़िम्मेदारियाँ दुष्टता 34,60,434 बननाये भूमिगत साइकिलवालों पेंशनयाता रूपचित्र उज़बेकिस्तान साथीसमर्थक उज़बेकिस्तान
Reference: अधिकरण में अपील करने के लिए कोई फीस नहीं देनी पड़ती

Source: शीर्षक कौन बनेगा करोड़पति (kaun banega crorepati)
Translation: ह्यजाऋचहृ परिशिष्ट (विचित्र) हिंसायाम त् एकजुटतायूरोपीय सीखीसुनी सुर्खदुख रायपुर तरकश सामानऋऊण्श्छष्यताया उपलइध zबात येश धनंजय तोललिंग कार्यसंगठन अधिवक़्ताओं ♫हे पर्यंत (विचित्र) खोजने बंदिश धनंजय एकजुटतायूरोपीय अनुवांशिकी ♫हे केलर (विचित्र) पाण्डुलिपियां कांग्रेर्सअधिवेशन साइज तोललिंग 3500 सैन्य, (विचित्र) सर्पदर्शभय सीढ़ीयों चारमीनार तुर्कमेनिस्तान मार्गशीर्ष निबटेंगे उठाना (नेपाली) ऊबड़खाबड़ मुखिया विजयादशमी उज़बेकिस्तान मॉडलर्स ऊबड़खाबड़
Reference: शीर्षक कौन बनेगा करोड़पति (kaun banega crorepati)

Transformers Model

Importing Necessary Libraries

```
import os
import pickle
import pandas as pd
import numpy as np
import random
import string
import re
import numpy as np
from nltk.translate.bleu_score import sentence_bleu
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau
from tensorflow.keras.layers import TextVectorization

import warnings
warnings.filterwarnings("ignore")
```

Initializing required Parameters and functions

```
batch_size = 64
embed_dim = 128
num_heads = 10
latent_dim = 2048
vocab_size = 20000
sequence_length = 20
dropout = 0.2

def preprocess_text(df):
    df["english_sentence"] = df["english_sentence"].apply(lambda x :
x.lower())
    df["hindi_sentence"] = df["hindi_sentence"].apply(lambda x :
x.lower())

    df["english_sentence"] = df["english_sentence"].apply(lambda x :
re.sub(r"http\S+", "", x))
    df["hindi_sentence"] = df["hindi_sentence"].apply(lambda x :
re.sub(r"http\S+", "", x))

    remove_digits = str.maketrans("", "",string.digits)
    df["english_sentence"] = df["english_sentence"].apply(lambda x :
x.translate(remove_digits))
    df["hindi_sentence"] = df["hindi_sentence"].apply(lambda x :
x.translate(remove_digits))
    df["hindi_sentence"] = df["hindi_sentence"].apply(lambda x :
re.sub("[a-zA-z२३०८९५७९४६]", "", x))

    special = set(string.punctuation)
```

```
    df['english_sentence'] = df['english_sentence'].apply(lambda x :
''.join(ch for ch in x if ch not in special))
    df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x :
''.join(ch for ch in x if ch not in special))

    df['english_sentence'] = df['english_sentence'].apply(lambda x:
re.sub("'", '', x))
    df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x:
re.sub("'", '', x))

    df['english_sentence'] = df['english_sentence'].apply(lambda x :
x.strip())
    df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x :
x.strip())
    df['english_sentence'] = df['english_sentence'].apply(lambda x :
re.sub(" +"," ",x))
    df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x :
re.sub(" +"," ",x))


    df["hindi_sentence"] = df["hindi_sentence"].apply(lambda x :
"[start] " + x + " [end]")

def translate(input_sentence):
    hindi_vocab = vec_hindi.get_vocabulary()
    hindi_index_lookup = dict(zip(range(len(hindi_vocab)),
hindi_vocab))
    max_decoded_sentence_length = 20

    tokenized_input_sentence = vec_english([input_sentence])
    decoded_sentence = "[start]"
    for i in range(max_decoded_sentence_length):
        tokenized_target_sentence = vec_hindi([decoded_sentence])
[:, :-1]
        predictions = transformer([tokenized_input_sentence,
tokenized_target_sentence])

        sampled_token_index = np.argmax(predictions[0, i, :])
        sampled_token = hindi_index_lookup[sampled_token_index]
        decoded_sentence += " " + sampled_token

        if sampled_token == "[end]":
            break

    return decoded_sentence[8:-5]

def format_dataset(eng, hin):
    eng = vec_english(eng)
    hindi = vec_hindi(hin)
    return ({"encoder_inputs" : eng, "decoder_inputs" : hindi[:, :-
```

```
1],}, hindi[:, 1:])


def to_dataset(df):
    dataset =
tf.data.Dataset.from_tensor_slices((df["english_sentence"].values,
df["hindi_sentence"].values))
    dataset = dataset.batch(batch_size)
    dataset = dataset.map(format_dataset)
    return dataset.shuffle(2048).prefetch(16).cache()
```

Preprocessing steps

```
df = pd.read_csv("Hindi_English_Truncated_Corpus(1).csv")
df.drop(["source"], axis=1, inplace = True)
df.dropna(axis = 0, inplace = True)

preprocess_text(df)

df.drop(df[df["english_sentence"] == " "].index, inplace = True)
df.drop(df[df["hindi_sentence"] == "[start]  [end]"].index, inplace =
True)

df["eng_sentence_length"] = df["english_sentence"].apply(lambda x :
len(x.split(' ')))
df["hindi_sentence_length"] = df["hindi_sentence"].apply(lambda x :
len(x.split(' ')))

df = df[df["eng_sentence_length"] <= 20]
df = df[df["hindi_sentence_length"] <= 20]

df = df.sample(n = 85000, random_state = 2048)
df = df.reset_index(drop = True)

train = df.iloc[:80000]
val = df.iloc[80000:84500]
test = df.iloc[84500:]

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


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

vec_english = TextVectorization(
    max_tokens = vocab_size, output_mode = "int",
output_sequence_length = sequence_length
```

```
    )

vec_hindi = TextVectorization(
    max_tokens = vocab_size, output_mode = "int",
output_sequence_length = sequence_length + 1,
standardize=custom_standardization
)

vec_english.adapt(df["english_sentence"].values)
vec_hindi.adapt(df["hindi_sentence"].values)
```

Saving the vectorizers for future use

```
pickle.dump({'config': vec_english.get_config(),
             'weights': vec_english.get_weights()}
            , open("vec_english.pkl", "wb"))

pickle.dump({'config': vec_hindi.get_config(),
             'weights': vec_hindi.get_weights()}
            , open("vec_hindi.pkl", "wb"))

import pickle
from tensorflow.keras.layers import TextVectorization

with open('vec_english.pkl', 'rb') as file:
    eng_data = pickle.load(file)

vec_english = TextVectorization.from_config(eng_data['config'])
vec_english.set_weights(eng_data['weights'])

with open('vec_hindi.pkl', 'rb') as file:
    hindi_data = pickle.load(file)

vec_hindi = TextVectorization.from_config(hindi_data['config'])
vec_hindi.set_weights(hindi_data['weights'])

train_ds = to_dataset(train)
val_ds = to_dataset(val)
```

Defining Model Architecture

```
class PositionalEmbedding(layers.Layer):
    def __init__(self, sequence_len, vocab_size, embed_dim, **kwargs):
        super(PositionalEmbedding, self).__init__(**kwargs)
        self.sequence_len = sequence_len
        self.vocab_size = vocab_size
        self.embed_dim = embed_dim
        self.token_embedding = layers.Embedding(
```

```
            input_dim = vocab_size, output_dim = embed_dim
        )
        self.position_embedding = layers.Embedding(
            input_dim = sequence_len, output_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_embedding(inputs)
        embedded_positions = self.position_embedding(positions)
        return embedded_tokens + embedded_positions

    def compute_mask(self, inputs, mask=None):
        return tf.math.not_equal(inputs, 0)

class TransformerEncoder(layers.Layer):
    def __init__(self, embed_dim, latent_dim, num_heads,
dropout,**kwargs):
        super(TransformerEncoder, self).__init__(**kwargs)
        self.embed_dim = embed_dim
        self.latent_dim = latent_dim
        self.num_heads = num_heads
        self.dropout = dropout
        self.attention = layers.MultiHeadAttention(
            num_heads = num_heads, key_dim = embed_dim
        )
        self.layer_norm1 = layers.LayerNormalization()
        self.layer_norm2 = layers.LayerNormalization()
        self.layer_ffn = keras.Sequential(
            [layers.Dense(latent_dim, activation="relu"),
             layers.Dropout(dropout),
             layers.Dense(embed_dim),]
            )
        self.supports_masking = True

    def call(self, inputs, mask = None):
        if mask is not None:
            padding_mask = tf.cast(mask[:, tf.newaxis, tf.newaxis, :],
dtype="int32")

        attention_output = self.attention(
            query = inputs, value = inputs, key = inputs,
attention_mask = padding_mask
        )
        ffn_input = self.layer_norm1(inputs + attention_output)
        ffn_output = self.layer_ffn(ffn_input)
        return self.layer_norm2(ffn_input + ffn_output)
```

```
class TransformerDecoder(layers.Layer):
    def __init__(self, embed_dim, latent_dim, num_heads,
sropout,**kwargs):
        super(TransformerDecoder, self).__init__(**kwargs)
        self.embed_dim = embed_dim
        self.latent_dim = latent_dim
        self.num_heads = num_heads
        self.dropout = dropout
        self.attention1 = layers.MultiHeadAttention(
            num_heads = num_heads, key_dim = embed_dim
        )
        self.attention2 = layers.MultiHeadAttention(
            num_heads = num_heads, key_dim = embed_dim
        )
        self.layer_ffn = keras.Sequential(
            [layers.Dense(latent_dim, activation="relu"),
             layers.Dropout(dropout),
             layers.Dense(embed_dim),]
        )
        self.layer_norm1 = layers.LayerNormalization()
        self.layer_norm2 = layers.LayerNormalization()
        self.layer_norm3 = layers.LayerNormalization()

        self.supports_masking = True

    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)

        attention_output1 = self.attention1(
            query=inputs, value=inputs, key=inputs,
attention_mask=causal_mask
        )
        out1 = self.layer_norm1(inputs + attention_output1)

        attention_output2 = self.attention2(
            query = out1, value = encoder_outputs, key =
encoder_outputs, attention_mask = padding_mask
        )
        out2 = self.layer_norm2(out1 + attention_output2)

        ffn_output = self.layer_ffn(out2)
        return self.layer_norm3(out2 + ffn_output)

    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)

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,
dropout,name="encoder_1")(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,
dropout,name="decoder_1")(x, encoded_seq_inputs)
x = layers.Dropout(0.4)(x)
decoder_outputs = layers.Dense(vocab_size, activation="softmax")(x)
decoder = keras.Model([decoder_inputs, encoded_seq_inputs],
decoder_outputs)

decoder_outputs = decoder([decoder_inputs, encoder_outputs])
transformer = keras.Model(
    [encoder_inputs, decoder_inputs], decoder_outputs,
name="transformer"
)
```

Compiling the model

```
early_stopping = EarlyStopping(patience = 5,restore_best_weights=True)

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
patience=3)

transformer.compile(
    optimizer = "adam",
    loss="sparse_categorical_crossentropy",
    metrics = ["accuracy"]
)
```

```
transformer.fit(train_ds, epochs = 50, validation_data = val_ds,
callbacks = [early_stopping, reduce_lr])
```

Saving weights for future use

```
transformer.save_weights("model.h5")

transformer.load_weights("model.h5")
```

Calculating the BLEU score

```
eng = test["english_sentence"].values
original = test["hindi_sentence"].values

translated = [translate(sent) for sent in eng]
bleu = 0

for i in range(test.shape[0]):
    bleu += sentence_bleu([original[i].split()],
translated[i].split(), weights = (0.5, 0.5))

print("BLEU score is : ", bleu / test.shape[0])
```

```
BLEU score is :  0.24585143664644363
```

Check the translations on some custom sentences

```
sentence_list = [
    "Hello World",
    "What are you doing?",
    "What time is it?",
    "Can you help me?",
    "This is an example of translation using transformers model"
]


for i in sentence_list:
    print("English Sentence : ",i)
    print("Translated Sentence : ",translate(i))
```

```
English Sentence :  Hello World
Translated Sentence :  दर्शक दुनिया
English Sentence :  What are you doing?
Translated Sentence :  आप क्या कर रहे हैं
English Sentence :  What time is it?
Translated Sentence :  तो क्या समय
English Sentence :  Can you help me?
Translated Sentence :  क्या आप मुझे मदद कर सकते हैं
```

```
English Sentence :  This is an example of translation using transformers model
Translated Sentence :  यह एक उदाहरण है जैसे भाषा का प्रयोग करता है
```