A

Project Report

on

English to Hindi translation using Attention Mechanism

Submitted in partial fulfillment of the requirements

for the award of the degree of

Bachelor of Technology

in

Data Science

by

Kavisha Jain (21520002)

Under the Supervision of

Mr Sukhdev Singh

JAGAN NATH UNIVERSITY, BAHADURGARH

BAHADURGARH-JHAJJAR ROAD, HARYANA

JAGAN NATH UNIVERSITY, BAHADURGARH
BAHADURGARH-JHAJJAR ROAD, HARYANA

CERTIFICATE

This is to certify that the project report entitled "English to hindi translation using Attention Mechanism" submitted by Ms. Kavisha Jain **,Roll No: 21520002** to the Jagan Nath University, Bahadurgarh Bahadurgarh-Jhajjar Road, Haryana in partial fulfillment for the award of Degree of Bachelor of Technology in Data Science is a bonafide record of the project work carried out by them under my supervision during the year 2024-2025.

Mr. Sukhdev Singh
(Project Supervisor)

ACKNOWLEDGEMENT

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend my sincere thanks to all of them.

We are highly indebted to Mr. Sukhdev Singh for his guidance and constant supervision. Also, we are highly thankful to them for providing necessary information regarding the project & also for their support in completing the project.

We are extremely indebted to Mr. Sukhdev Singh, Department of Information Technology, for their valuable suggestions and constant support throughout my project tenure. We would also like to express our sincere thanks to all faculty and staff members of the Department for their support in completing this project on time.

We also express gratitude towards our parents for their kind cooperation and encouragement which helped me in completion of this project. Our thanks and appreciation also go to our friends in developing the project and all the people who have willingly helped me out with their abilities.

(Kavisha Jain)

ABSTRACT

Language translation bridges communication gaps, enabling people to share information and foster relationships globally. In today's interconnected world, the digital era has transformed how we communicate, share information, and conduct business. At the heart of this transformation lies language translation, a critical tool that breaks down linguistic barriers and fosters global connectivity. The importance of language translation in the digital era cannot be overstated, as it plays a pivotal role in various aspects of our lives, from personal communication to international business and beyond.Neural Machine Translation (NMT) has revolutionized this process by providing automated and accurate translations. This project investigates the effectiveness of NMT models enhanced by attention mechanisms for translating between English and Hindi

This project focuses on developing an automated translation system from English to Hindi using advanced deep learning techniques. The system leverages Sequence-to-Sequence (Seq2Seq) models, attention mechanisms, and to enhance translation accuracy and efficiency. The project aims to overcome the limitations of traditional translation methods and provide robust, high-quality translations.

Keywords:Language translation Model ,Neural Machine Translation (NMT), Sequence-to-Sequence (Seq2Seq) models, attention mechanisms,

ABBREVIATIONS

ML - Machine Learning
DL-Deep learning
NLP - Natural Language Processing
NMT-Neural Machine Translation
NLTK - Natural language toolkit

INDEX

CHAPTER 1:INTRODUCTION

Translation includes more than simply translating words from one language to another. It builds bridges between cultures and helps people to communicate better. However, skilled translators are needed to do this who have a good knowledge of both the languages. Often, meanings can be misinterpreted. We require a system that accurately translates between languages and is less prone to human error.We have developed a translation model which converts text from English to Hindi. We have selected Hindi because it is the native language of around 260 million people worldwide. Not everyone in India knows English fluently, and tourists from other countries might struggle to express themselves. Hindi has become essential for organizations who wish to develop and expand their business in India. Various books and movies about our culture are written in English and can be successful around the world if translated accurately

In today's interconnected digital era, language translation plays a crucial role in facilitating communication, business, and cultural exchange. It bridges communication gaps among people speaking over 7,000 different languages, enabling effective interaction through social media, online forums, and instant messaging. For businesses, accurate translation of documents and marketing materials is essential for smooth international operations and building trust. Translation democratizes access to information and education, allowing non-native speakers to benefit from global knowledge resources. The travel and tourism industry also thrives on translation services that help travelers navigate foreign environments. In multicultural societies, translation ensures equal access to essential services, fostering inclusivity and social cohesion. Technological advancements in machine translation, powered by artificial intelligence, continue to enhance accuracy and efficiency, making real-time translation increasingly possible. Overall, language translation is indispensable in overcoming linguistic barriers, promoting global connectivity, and ensuring seamless communication in a diverse world.

The purpose of this project is to enhance the translation of English to Hindi using neural networks, specifically focusing on the application of attention mechanisms in Neural Machine Translation (NMT). As language translation is pivotal for global communication, business, and cultural exchange, improving the accuracy and fluency of translations between widely spoken languages like English and Hindi is crucial.

Attention mechanisms have emerged as a significant advancement in the field of NMT. Traditional sequence-to-sequence models often struggle with long sentences and complex syntactic structures. Attention mechanisms address these challenges by allowing the model to focus on specific parts of the input sequence, thereby improving the handling of long-distance dependencies and enhancing the overall contextual understanding. This results in more accurate and fluent translations, making attention mechanisms indispensable for high-quality NMT systems.

In today's globalized world, language translation plays a crucial role in bridging communication gaps, facilitating international business, and making information accessible across linguistic boundaries. Neural Machine Translation (NMT) has revolutionized the field of automated translation, offering significant improvements over traditional rule-based and statistical methods.

Developing an effective English to Hindi translation system poses several challenges, including handling long-range dependencies, maintaining context, and accurately translating idiomatic expressions. Traditional methods fail to capture the complexities of languages, resulting in poor translation quality.This project aims to develop an English to Hindi translation system using deep learning techniques, including Seq2Seq models, attention mechanisms, and Transformers, to achieve accurate and efficient translations.

1.2 Description of Theoretical concepts

In the realm of machine learning (ML) and deep learning (DL), efficient data manipulation is paramount for handling large-scale datasets and performing complex computations. This introduction delves into the crucial roles played by key Python libraries—NumPy, Pandas, Matplotlib, Seaborn, and NLTK—along with the utilization of Google Colab as the development platform.

Emerging Trends in Translation Models

1. Transformer Models: The introduction of Transformer models, such as BERT, GPT, and T5, has revolutionized NMT. These models leverage self-attention mechanisms to process entire sentences simultaneously, improving translation accuracy and efficiency.

2. Multilingual Models: Recent advances have led to the development of multilingual translation models that can handle multiple language pairs with a single model. Examples include Facebook's M2M-100 and Google's Multilingual Neural Machine Translation (MNMT).

3. Pre-trained Language Models: Pre-trained language models, fine-tuned on specific translation tasks, have shown remarkable improvements in translation quality. Models like OpenAI's GPT-3 and Google's T5 demonstrate the potential of transfer learning in NMT.

4. Hybrid Approaches: Combining the strengths of different translation models, hybrid approaches aim to enhance translation quality. For example, integrating rule-based corrections with neural models can address specific linguistic challenges.

Types of Translation Models in Data Science

Translation models are crucial in breaking down language barriers and facilitating communication in a globalized world. The primary types of translation models include Rule-Based Machine Translation (RBMT), Statistical Machine Translation (SMT), and Neural Machine Translation (NMT). Each type has distinct methodologies, advantages, and limitations.

1. Rule-Based Machine Translation (RBMT)

Methodology: RBMT relies on linguistic rules and dictionaries to translate text. It uses a combination of syntactic, morphological, and semantic analysis to convert sentences from the source language to the target language.

Advantages

High Control and Predictability: The translation process is transparent, allowing linguists to tweak and refine the rules for better accuracy.

Consistency: Since the translation rules are explicitly defined, the output is consistent for similar inputs.

Limitations

Scalability Issues: Developing and maintaining comprehensive linguistic rules for each language pair is labor-intensive and time-consuming.

Inflexibility: The system may struggle with idiomatic expressions, slang, and evolving language use.

2. Statistical Machine Translation (SMT)

Methodology: SMT uses statistical models derived from bilingual text corpora to predict the probability of a sequence of words in the target language given a sequence in the source language. The most common approach within SMT is the phrase-based model, which translates text by considering sequences of words (phrases) rather than individual words.

Advantages

Data-Driven: SMT can improve over time as more bilingual data becomes available.

Flexible: Capable of handling various language pairs without extensive manual intervention.

Limitations

Quality of Data Dependency: The quality of the translation heavily depends on the size and quality of the training data.

Contextual Errors: SMT may struggle with long-range dependencies and context, leading to inaccuracies in translation.

3. Neural Machine Translation (NMT)

Methodology: NMT utilizes deep learning techniques, specifically neural networks, to perform translation. The most common architecture for NMT is the sequence-to-sequence (seq2seq) model, often enhanced with attention mechanisms to better handle long sentences and maintain context.

Emerging Trends in Translation Models

1. Transformer Models: The introduction of Transformer models, such as BERT, GPT, and T5, has revolutionized NMT. These models leverage self-attention mechanisms to process entire sentences simultaneously, improving translation accuracy and efficiency.

2. Multilingual Models: Recent advances have led to the development of multilingual translation models that can handle multiple language pairs with a single model. Examples include Facebook's M2M-100 and Google's Multilingual Neural Machine Translation (MNMT).

3. Pre-trained Language Models: Pre-trained language models, fine-tuned on specific translation tasks, have shown remarkable improvements in translation quality. Models like OpenAI's GPT-3 and Google's T5 demonstrate the potential of transfer learning in NMT.

4. Hybrid Approaches: Combining the strengths of different translation models, hybrid approaches aim to enhance translation quality. For example, integrating rule-based corrections with neural models can address specific linguistic challenges.

Attention Mechanism in Translation

The field of machine translation has seen remarkable advancements over the years, with attention mechanisms playing a pivotal role in improving the quality and accuracy of translations. This introduction explores the concept of attention mechanisms in translation models, highlighting their significance and impact on neural machine translation (NMT).

Understanding Neural Machine Translation (NMT)

Neural Machine Translation (NMT) models represent the cutting-edge approach in the domain of automated translation. Unlike traditional rule-based or statistical methods, NMT leverages deep learning techniques to model the translation process. These models typically employ encoder-decoder architectures, where the encoder processes the input sentence (source language) and the decoder generates the translated output sentence (target language). Despite their effectiveness, early NMT models faced challenges in handling long sentences and capturing contextual dependencies effectively.

Benefits of Attention Mechanisms

The introduction of attention mechanisms in translation models brings several key benefits:

1. **Enhanced Contextual Understanding:** By focusing on different parts of the input sequence for each word in the output, attention mechanisms enable models to capture contextual nuances more effectively. This results in translations that are more coherent and contextually accurate.

2. **Improved Handling of Long Sentences:** Traditional NMT models struggled with long sentences due to the fixed-length encoding bottleneck. Attention mechanisms alleviate this issue by allowing the model to consider all input words dynamically, regardless of the sentence length.
3. **Alignment Interpretability:** Attention weights provide insights into how the model aligns words between the source and target languages. This alignment information can be visualized, offering interpretability and helping to understand the translation process better.
4. **Increased Flexibility:** Attention mechanisms make it easier to handle various translation tasks, including multilingual translation and domain-specific adaptations. The ability to focus on relevant parts of the input sequence makes the model adaptable to different linguistic and contextual requirements.

Applications Beyond Translation

While attention mechanisms have revolutionized machine translation, their applications extend beyond this domain. They are widely used in various natural language processing (NLP) tasks, such as text summarization, question answering, and sentiment analysis. Additionally, attention mechanisms have found applications in computer vision tasks, such as image captioning and object detection, further showcasing their versatility and impact.

Attention mechanisms have emerged as a cornerstone in the development of advanced neural machine translation models. By enabling dynamic focus on relevant parts of the input sequence, they significantly enhance the model's contextual understanding, handling of long sentences, and overall translation quality. As the field of machine translation continues to evolve, attention mechanisms will remain a critical component in achieving more accurate and contextually aware translations, ultimately bridging language barriers more effectively.

MOTIVATION

The motivation for this project stems from the increasing need for accurate and efficient language translation in our globally connected world. As English and Hindi are among the most widely spoken languages, the ability to translate between them with high accuracy has significant implications for various sectors, including business, education, tourism, and entertainment. Language barriers often hinder effective communication and collaboration across borders. By improving English-to-Hindi translation, we can facilitate better understanding and cooperation among individuals and organizations, fostering global connectivity and cultural exchange.

As companies expand their operations internationally, the ability to communicate effectively with partners, clients, and customers who speak different languages becomes essential. High-quality translation ensures that businesses can operate smoothly in diverse linguistic markets, enhancing customer satisfaction, building trust, and driving growth. Additionally, the digital era has democratized access to knowledge and information. Accurate translation of educational resources, research papers, and online content into Hindi allows non-native English speakers to benefit from a wealth of global information, promoting education and lifelong learning.

Advancements in artificial intelligence (AI) and machine learning (ML) have opened new possibilities for improving translation quality. Attention mechanisms, in particular, have shown great promise in enhancing neural machine translation (NMT) models. By leveraging these innovations, we aim to push the boundaries of what is achievable in automated language translation. Improved English-to-Hindi translation has practical applications in many fields. For instance, in the tourism industry, it helps travelers navigate foreign environments and communicate with locals. In healthcare, it ensures that patients receive accurate information regardless of their language. In government and legal services, it promotes inclusivity and equal access to essential information and services.

Overall, this project is driven by the potential to make a meaningful impact on global communication and understanding. By harnessing the power of attention mechanisms in NMT, we aim to create a robust solution for English-to-Hindi translation that meets the demands of the modern, interconnected world.

CHAPTER 2: LITERATURE REVIEW

Machine translation (MT) has significantly evolved since its inception in the 1950s, with the primary goal of overcoming linguistic barriers through automated translation systems. In today's interconnected world, the importance of MT has grown, driven by the need for accurate translation of scientific research, technical documents, and everyday communication across different languages. This review delves into the various MT approaches, particularly the neural machine translation (NMT) using sequence-to-sequence (Seq2seq) models, highlighting their methodologies, advantages, and limitations.

MT research began with rule-based approaches that relied on linguistic rules crafted by human experts. These systems, known as rule-based machine translation (RBMT), determine how words and phrases should be translated in different contexts. The primary advantage of RBMT is its complete control over translation rules, which can be reused across different language pairs. However, it requires extensive dictionaries and linguistic expertise, making it less adaptable to new languages and contexts without significant manual intervention. Over time, statistical machine translation (SMT) emerged, utilizing bilingual text corpora to translate phrases rather than individual words, thus improving the contextual accuracy of translations. These systems handle phrase-based translations more effectively, addressing some limitations of word-based translations. However, SMT still struggles with long-distance dependencies and requires extensive parallel corpora, which may not always be available for all language pairs.

NMT represents a significant leap in MT technology, utilizing deep learning techniques to perform end-to-end translation. The Seq2seq model with attention mechanisms, introduced by Google in 2014, has become a cornerstone in NMT. This model consists of an encoder that processes the input sentence and a decoder that generates the translated output. The attention mechanism allows the model to focus on relevant parts of the input sentence during translation, improving accuracy, especially for long sentences. The Seq2seq model operates through several key steps: segmentation, where the input text is tokenized into manageable units or tokens; tagging, where proper nouns and numbers are tagged with placeholders to avoid erroneous translations; translation, where the segmented and tagged tokens from the source language (English) are converted to the target language (Hindi); and rearranging, where the translated tokens are rearranged to form coherent sentences in the target language. The model is trained on a diverse dataset, such as the "English to Hindi Neural Machine Translation" dataset from Kaggle, to enhance its accuracy and ability to handle various linguistic nuances.

NMT systems, particularly those utilizing LSTM (Long Short-Term Memory) networks, offer several advantages over traditional MT approaches. They can process and translate large volumes of text quickly without human intervention, remember and utilize context over longer text sequences, and improve over time with additional data and training. However, challenges remain, such as the need for extensive computational resources for training and the difficulty in handling highly complex sentence structures or idiomatic expressions. While RNNs (Recurrent Neural Networks) and LSTMs have significantly advanced the capabilities of NMT systems, they also come with their own set of drawbacks. RNNs suffer from the vanishing gradient problem, where gradients used for training the network diminish exponentially as they are propagated back through time, making it difficult for the network to learn long-term dependencies. Additionally, RNNs struggle with retaining information over long sequences, which can impact the accuracy of translations involving long sentences or documents. LSTMs, while designed to mitigate the vanishing gradient problem, are more complex and computationally expensive to train compared to simple RNNs. This can be a significant drawback when dealing with very large datasets or when computational resources are limited. Although LSTMs perform better than RNNs in handling long-term dependencies, they can still face challenges with very long sequences, where the context may extend beyond the network's capacity to remember.

The evolution of MT from RBMT and SMT to NMT has brought significant improvements in translation quality and efficiency. The Seq2seq model, with its encoder-decoder architecture and attention mechanisms, represents the current state-of-the-art in NMT, capable of producing highly accurate translations between languages. Despite its advantages, ongoing research and development are necessary to address the remaining challenges and further enhance the capabilities of MT systems. By understanding the progression and methodologies in machine translation, researchers and practitioners can continue to innovate and improve the technology, making it more accessible and effective for diverse linguistic needs worldwide.

CHAPTER 3: PROBLEM FORMULATION

Problem Statement

Objective: To develop an efficient Neural Machine Translation (NMT) system that translates English sentences into Hindi using an attention mechanism to improve translation quality and handle long-distance dependencies.

Theoretical concepts used

NumPy: Efficient Data Handling

NumPy is designed for efficiently managing large multidimensional arrays, crucial for machine learning (ML) and deep learning (DL). It provides powerful array operations, including element-wise arithmetic, broadcasting, and slicing, which simplify complex matrix computations in models like linear regression and neural networks. NumPy's efficiency in performing these operations makes it essential for handling vast datasets and speeding up training and inference times. It also aids in data preprocessing tasks like normalization and feature extraction, ensuring data is in a suitable format for models.

Pandas: Structured Data Analysis

Pandas excels in data preprocessing, handling missing values, duplicates, and outliers, which are common in real-world datasets. It simplifies feature engineering, allowing the creation and combination of features and encoding categorical variables, enhancing model predictive power. Pandas facilitates data splitting into training, validation, and test sets, crucial for preventing overfitting. It streamlines exploratory data analysis (EDA) by summarizing data, calculating statistics, and visualizing distributions, guiding feature engineering and model-building processes.

Matplotlib: Visualization and Model Evaluation

Matplotlib is a versatile library for data visualization, offering a range of plots like line charts, scatter plots, histograms, and heatmaps. It is essential for visualizing data distributions, feature importance, and model performance metrics like accuracy, precision, recall, and ROC curves. Matplotlib aids in creating confusion matrices and visualizing decision boundaries, helping interpret model predictions and improvements. In NLP, it generates word clouds and visualizes hyperparameter search results, aiding in model optimization.

Seaborn: Advanced Visualization

Seaborn, built on Matplotlib, offers advanced and aesthetically pleasing visualizations. It simplifies creating complex plots like violin plots, pair plots, and heatmaps, providing deeper insights into data. Seaborn's themes and color palettes improve plot aesthetics, making them more appealing and easier to interpret.

NLTK: Natural Language Processing

The Natural Language Toolkit (NLTK) is a comprehensive suite for text processing in Python. It supports tasks like stemming, tokenization, and removing stop words, essential for text preprocessing. NLTK offers libraries for text classification, clustering, and topic modeling, making it versatile for various NLP applications. It helps build models that classify text, cluster documents, and extract topics, essential for NLP research and practice.

.

Google Colab: Development Platform

For this project, Google Colab is employed as the development platform. It is a free, cloud-based IDE that simplifies code sharing through Google Drive integration. Colab comes preinstalled with many frequently used modules and features a user-friendly interface, supporting even Anaconda IDE. This makes it an ideal platform for collaborative ML/DL development and experimentation.

Google Colab offers the advantage of powerful computational resources, including GPU and TPU support, which are essential for training large ML/DL models. By leveraging these resources, developers can significantly speed up the training process and handle larger datasets more effectively.

Colab's collaborative features allow multiple users to work on the same notebook simultaneously, making it easy to share code, insights, and results. This fosters collaboration and knowledge sharing among team members, enhancing productivity and accelerating project development.

Deep learning and Natural Language Processing (NLP) are at the heart of our major project focused on English to Hindi translation. By leveraging these advanced technologies, we aim to develop a highly accurate and efficient translation system. Below, we detail how deep learning and NLP are integrated into our project code.

Seq2Seq models are foundational to our translation system. These models consist of two main components:

1. **Encoder**: This component processes the input English sentence and encodes it into a fixed-size context vector. We utilize Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks as the underlying architectures for our encoders.
2. **Decoder**: Using the context vector generated by the encoder, the decoder generates the corresponding Hindi translation. The decoder is also implemented using LSTM and GRU networks.

In our code, we define and train these Seq2Seq models using TensorFlow and Keras, which provide the necessary tools and abstractions for building and training deep learning models.

Attention Mechanisms

Attention mechanisms enhance the capability of Seq2Seq models by allowing them to focus on different parts of the input sentence when generating each word of the translation. This is particularly important for handling long-range dependencies and improving translation accuracy.

. The attention mechanism computes attention scores, which are used to create a weighted context vector from the encoder's outputs. This context vector is then fed into the decoder along with the current decoding state, enabling the model to make more informed predictions.

The implementation involves calculating attention weights and incorporating them into the model architecture. The attention layer is integrated into the TensorFlow/Keras model, ensuring that the model dynamically attends to relevant parts of the input during training and inference.

Benefits of Attention Mechanism

The attention mechanism offers several benefits in neural machine translation (NMT), significantly improving translation quality. One of the key advantages is its ability to handle long sentences effectively. Traditional encoder-decoder models without attention often struggle with long sentences as they compress the entire input sentence into a single fixed-length context vector. In contrast, the attention mechanism allows the decoder to access the entire sequence of encoder hidden states, enabling it to focus on the most relevant parts of the input sentence at each decoding step. For example, without attention, translating a sentence like "The quick brown fox jumps over the lazy dog" might result in losing important details towards the end of the sentence. With attention, the model can dynamically focus on "fox" when translating "jumps" and "dog" when translating "over," ensuring that the meaning is preserved throughout the translation process.

The attention mechanism also excels in handling complex sentence structures. Sentences with multiple clauses, subclauses, and nuanced meanings benefit significantly as the model can allocate attention to different parts of the input as needed, capturing intricate relationships between words and phrases more effectively. For instance, when translating "Although it was raining, she went for a walk," the model can focus on "raining" when translating "although" and "walk" when translating "went for," ensuring that the cause-effect relationship is correctly translated.

Moreover, the attention mechanism improves word alignment between source and target languages. It provides a way to align words in the source sentence with their corresponding words in the target sentence more accurately. During translation, alignment scores help identify which words in the source sentence are most relevant to the current word being generated in the target sentence. These scores are normalized to produce attention weights, indicating the degree of focus on each word in the source sentence. For example, when translating the French sentence "Le chat noir dort sur le canapé" to English, the model can focus on "chat" for "cat" and "noir" for "black," ensuring accurate word alignment.

CHAPTER 4: SYSTEM DESIGN

1.Data Loading: Pandas is used to load and preprocess the data. For instance,

reading the dataset from a CSV file from kaggle.

Parallel Corpus: Collect a bilingual corpus containing pairs of English sentences and their corresponding Hindi translations. Eg IIT Bombay English-Hindi dataset

2.Data Cleaning: Handling missing values, filtering, and transforming data.

3.Exploratory Data Analysis (EDA): Analyzing the dataset to understand the distribution and relationships between different variables.

4.Preprocessing the data frame:

Text Cleaning: Remove any unwanted characters, HTML tags, and punctuation.

Tokenization: Split sentences into tokens (words).

Lowercasing: Convert all text to lowercase to maintain uniformity.

Padding: Pad sentences to ensure uniform length across the dataset.

5.Attention Mechanism Model

The architecture of the Neural Machine Translation (NMT) system consists of three main components: the Encoder, the Attention Mechanism, and the Decoder. These components collaborate to translate input sentences from English to Hindi effectively.

1. Encoder: The encoder is responsible for processing the input sentence and converting it into a set of feature vectors (context vectors), which are then used by the decoder to generate the output sentence. The encoder consists of the following layers:

Input Layer: The input layer takes the tokenized and padded input sentence. The input sentence is represented as a sequence of integers, where each integer corresponds to a word in the sentence.

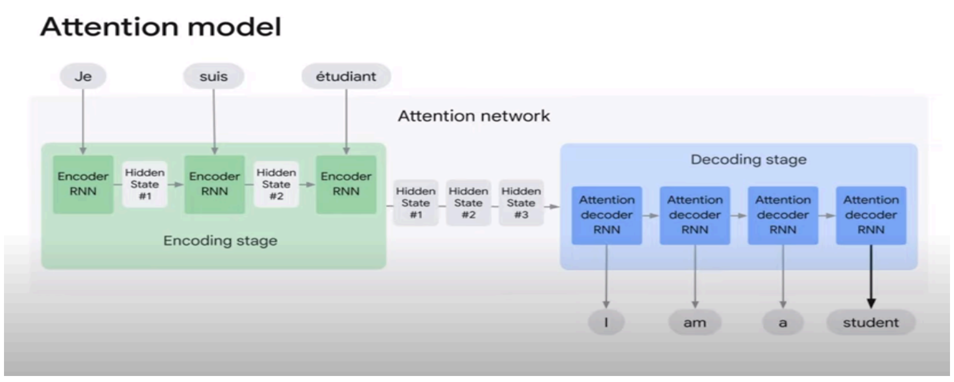
Embedding Layer: The embedding layer converts the input tokens into dense vectors of a fixed size. This transformation helps in capturing the semantic meaning of the words in a continuous vector space.

Bidirectional Long Short-Term Memory (LSTM) Network: The bidirectional LSTM processes the embedded input sequence in both forward and backward directions. This bidirectional approach allows the encoder to capture context from both past and future words in the sentence, enhancing the quality of the generated feature vectors. The LSTM outputs a sequence of feature vectors (one for each word) along with the final hidden and cell states for both directions.

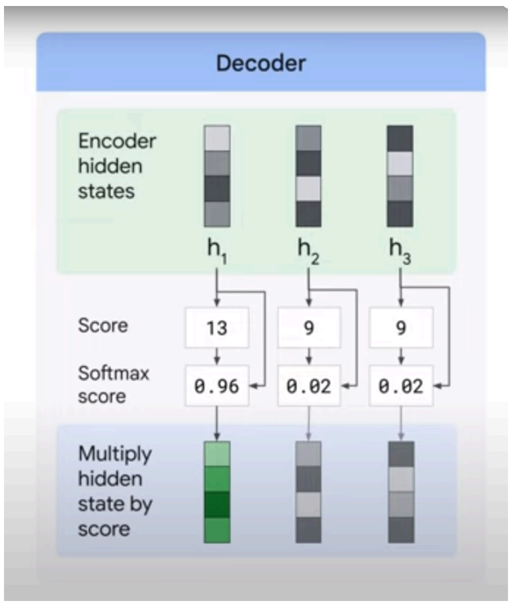
2. Attention Mechanism

The attention mechanism plays a crucial role in improving translation quality by allowing the decoder to focus on different parts of the input sentence at each step of the output generation. The attention mechanism computes a weighted sum of the encoder outputs (context vectors) based on the current state of the decoder. This weighted sum, known as the context vector, provides relevant information to the decoder for generating the next word in the output sentence.

Attention Layer: The attention layer calculates the attention weights, which determine the importance of each word in the input sentence for generating the current word in the output sentence. The context vector is computed as a weighted sum of the encoder outputs, where the weights are the attention scores.



3. Decoder



The decoder generates the output sentence (Hindi) one word at a time, using the context vector and the previous hidden state to produce each word. The decoder consists of the following layers:

Input Layer: The input layer takes the previously generated word (or a start token at the beginning) as input for the next time step.

Embedding Layer: Similar to the encoder, the embedding layer in the decoder converts the input tokens into dense vectors.

LSTM Network: The LSTM network in the decoder processes the embedded input token and the context vector from the attention mechanism to generate the next hidden state. The LSTM outputs a sequence of vectors, one for each word in the output sentence.

Dense Layer: The dense layer with a softmax activation function generates the probability distribution over the target vocabulary for the next word. The word with the highest probability is selected as the next word in the output sentence.

Training and Inference : During training, the model uses teacher forcing, where the actual target word is provided as the next input to the decoder, helping the model learn the correct sequence of words. The model is trained using the categorical cross-entropy loss function and optimized using the Adam optimizer.

During inference, the decoder generates the output sentence one word at a time, using the previously generated word as input for the next step. The process continues until the end-of-sentence token is generated or a maximum length is reached.

Attention Visualization

The attention weights can be visualized to understand which parts of the input sentence the model focuses on while generating each word of the output sentence. This visualization provides insights into the model's translation process and highlights the importance of the attention mechanism.

BLEU Score

BLEU (BiLingual Evaluation Understudy) is a metric for automatically evaluating machine-translated text. The BLEU score is a number between zero and one that measures the similarity of the machine-translated text to a set of high quality reference translations. A value of 0 means that the machine-translated output has no overlap with the reference translation (low quality) while a value of 1 means there is perfect overlap with the reference translations (high quality).

It has been shown that BLEU scores correlate well with human judgment of translation quality. Note that even human translators do not achieve a perfect score of 1.0.

Example Calculation:

Suppose we have a reference sentence: “The cat is on the mat.”

And a candidate translation: “The the the cat mat.”

BLEU score calculation:

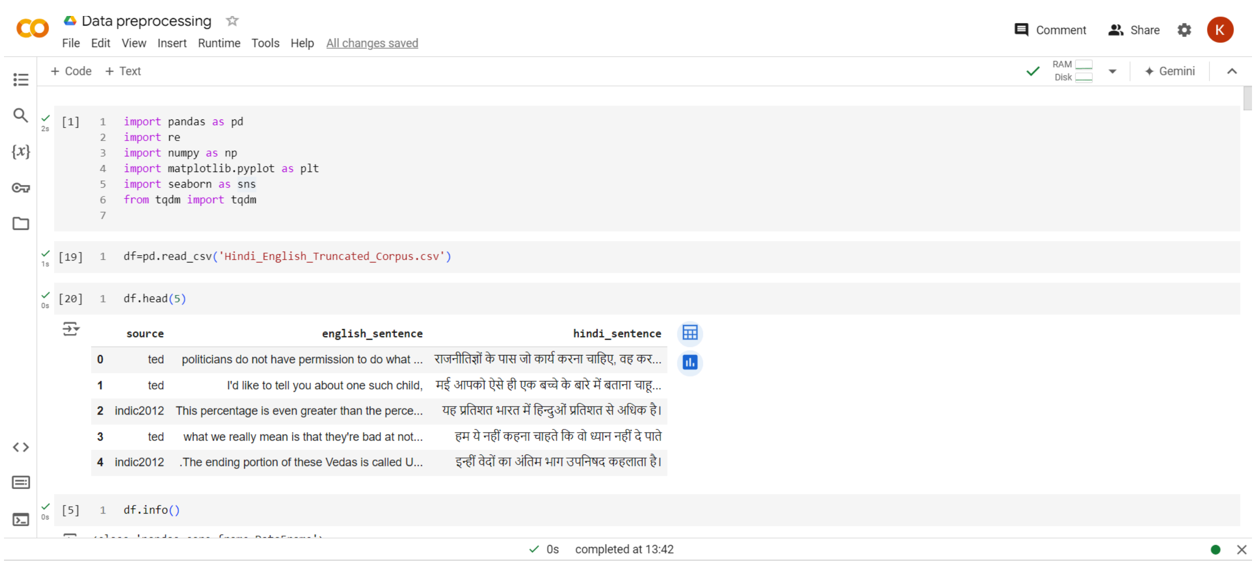
Unigrams (1-grams) match: “the,” “cat,” “mat” (3 matches).

Bigrams (2-grams) match: “the cat,” “cat mat” (2 matches).

Total matches = 5.

BLEU score = 5/6 ≈ 0.83 (weighted by n-gram length).

CHAPTER 5: IMPLEMENTATION



```
[1]
import pandas as pd
import re
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
```

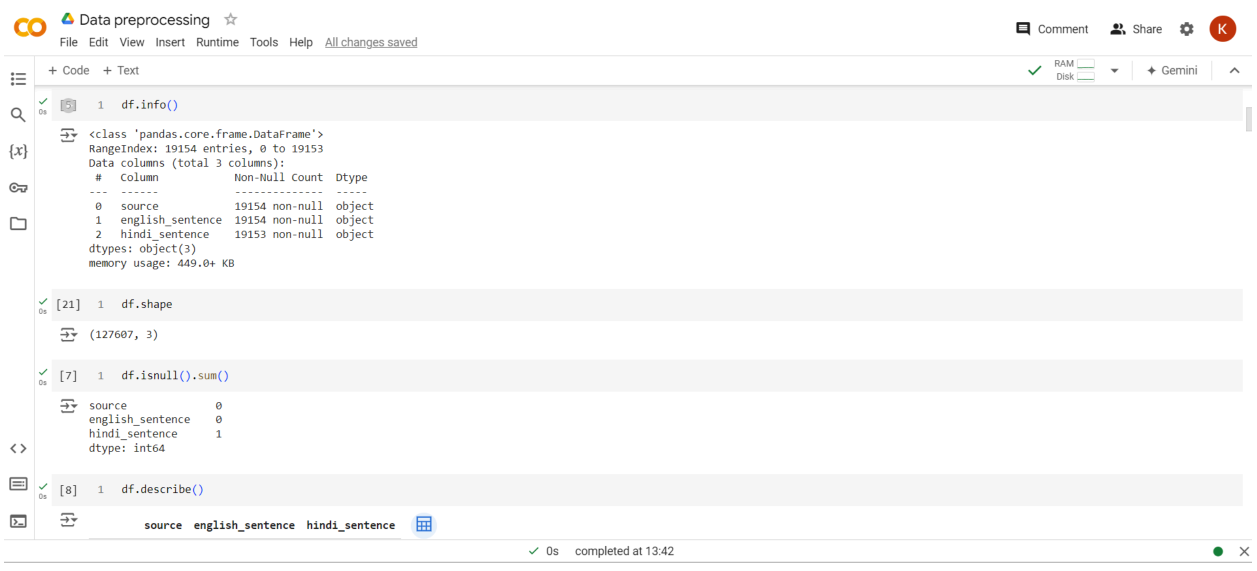
```
[19]
df=pd.read_csv('Hindi_English_Truncated_Corpus.csv')
```

```
[20]
df.head(5)
```

	source	english_sentence	hindi_sentence
0	ted	politicians do not have permission to do what ...	राजनीतिज्ञों के पास जो कार्य करना चाहिए, वह कर...
1	ted	I'd like to tell you about one such child,	मई आपको ऐसे ही एक बच्चे के बारे में बताना चाहू...
2	indic2012	This percentage is even greater than the perce...	यह प्रतिशत भारत में हिन्दुओं प्रतिशत से अधिक है।
3	ted	what we really mean is that they're bad at not...	हम ये नहीं कहना चाहते कि वो ध्यान नहीं दे पाते
4	indic2012	.The ending portion of these Vedas is called U...	इन्हीं वेदों का अंतिम भाग उपनिषद कहलाता है।

```
[5]
df.info()
```

0s completed at 13:42



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19154 entries, 0 to 19153
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   source            19154 non-null  object
 1   english_sentence  19154 non-null  object
 2   hindi_sentence    19153 non-null  object
dtypes: object(3)
memory usage: 449.0+ KB
```

```
[21]
df.shape
```

```
(127607, 3)
```

```
[7]
df.isnull().sum()
```

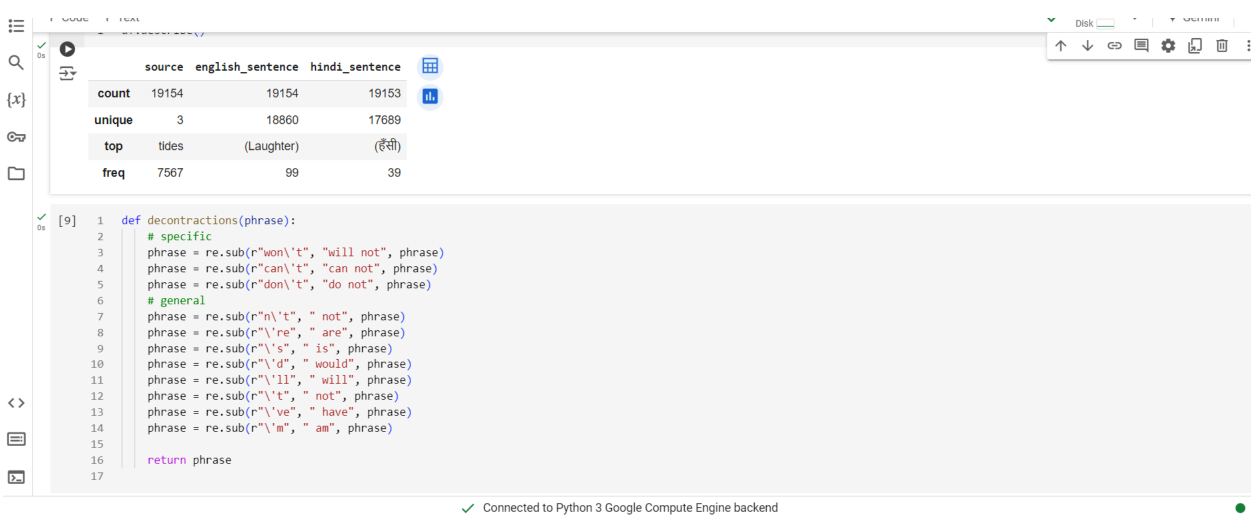
```
source              0
english_sentence    0
hindi_sentence      1
dtype: int64
```

```
[8]
df.describe()
```

```
      source  english_sentence  hindi_sentence
```

0s completed at 13:42

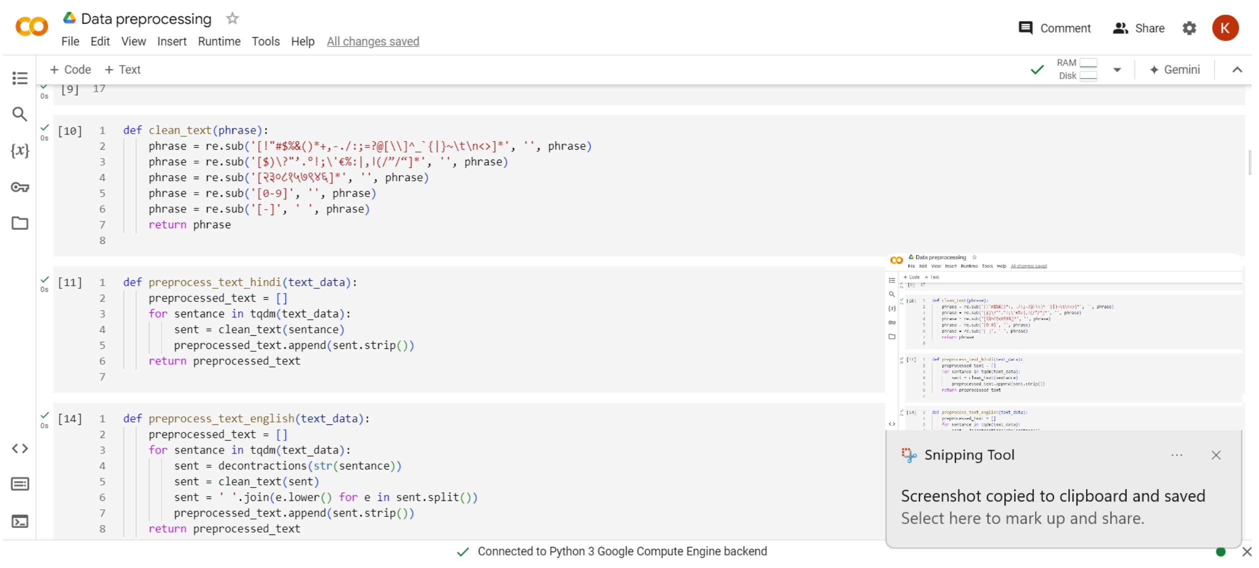
DATA PREPROCESSING



	source	english_sentence	hindi_sentence
count	19154	19154	19153
unique	3	18860	17689
top	tides	(Laughter)	(हँसी)
freq	7567	99	39

```
[9]
def decontractions(phrase):
    # specific
    phrase = re.sub(r"won\'t", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    phrase = re.sub(r"don\'t", "do not", phrase)
    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)

    return phrase
```

Connected to Python 3 Google Compute Engine backend



```
[9] 17
```

```
[10]
def clean_text(phrase):
    phrase = re.sub('[!"#$%&()*+,-./:;=?@[\\]^_`{|}~\t\n<>]*', '', phrase)
    phrase = re.sub('[$)\?"’.°!;\'€%:|,।(/”/“]*', '', phrase)
    phrase = re.sub('[२३०८१५७९४६]*', '', phrase)
    phrase = re.sub('[0-9]', '', phrase)
    phrase = re.sub('[-]', ' ', phrase)
    return phrase
```

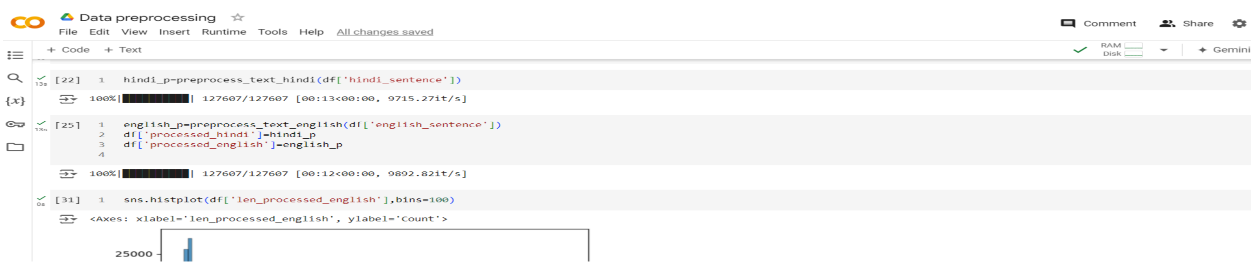
```
[11]
def preprocess_text_hindi(text_data):
    preprocessed_text = []
    for sentance in tqdm(text_data):
        sent = clean_text(sentance)
        preprocessed_text.append(sent.strip())
    return preprocessed_text
```

```
[14]
def preprocess_text_english(text_data):
    preprocessed_text = []
    for sentance in tqdm(text_data):
        sent = decontractions(str(sentance))
        sent = clean_text(sent)
        sent = ' '.join(e.lower() for e in sent.split())
        preprocessed_text.append(sent.strip())
    return preprocessed_text
```

Snipping Tool

Screenshot copied to clipboard and saved
Select here to mark up and share.

Connected to Python 3 Google Compute Engine backend



```
[22]
hindi_p=preprocess_text_hindi(df['hindi_sentence'])
```

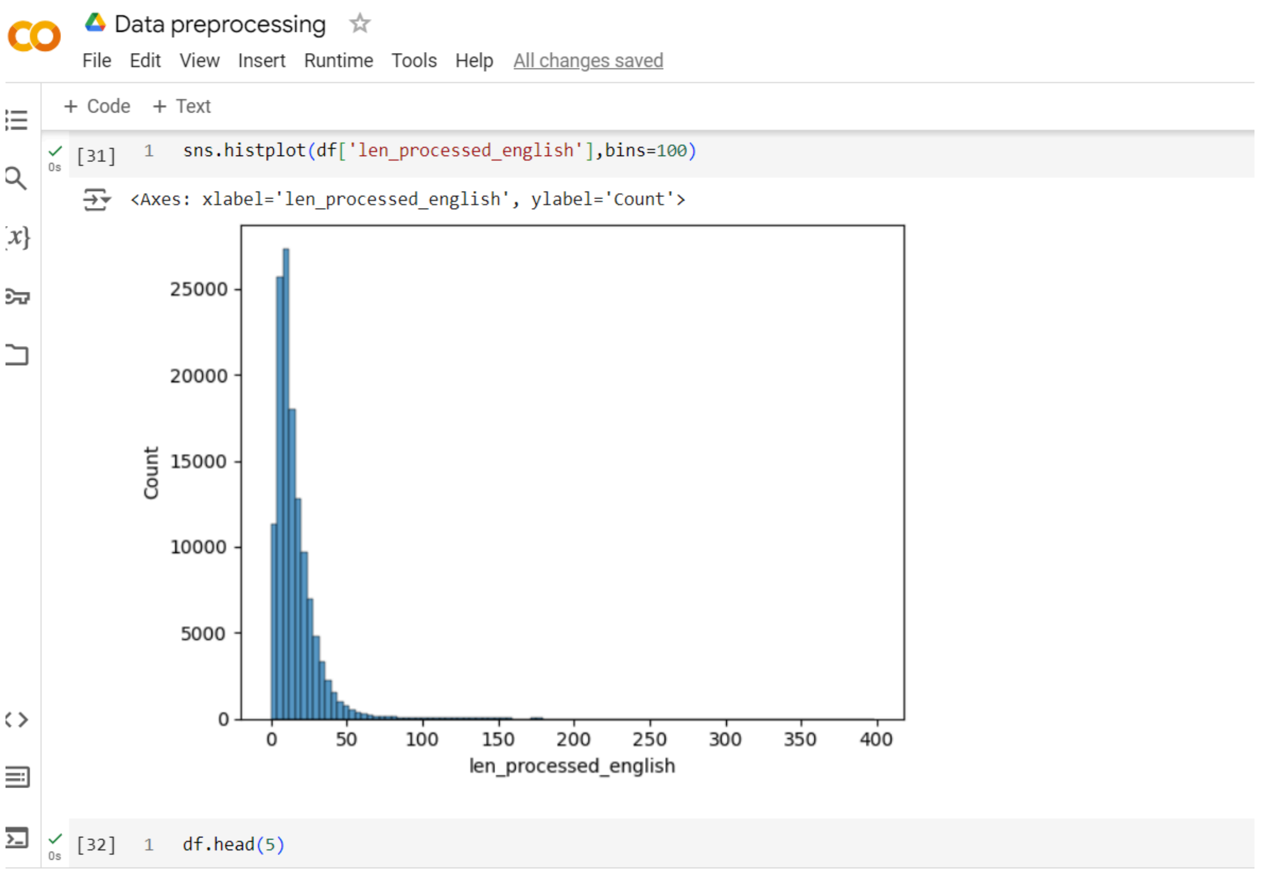
```
100%|██████████| 127607/127607 [00:13<00:00, 9715.27it/s]
```

```
[25]
english_p=preprocess_text_english(df['english_sentence'])
df['processed_hindi']=hindi_p
df['processed_english']=english_p
```

```
100%|██████████| 127607/127607 [00:12<00:00, 9892.82it/s]
```

```
[31]
sns.histplot(df['len_processed_english'],bins=100)
```

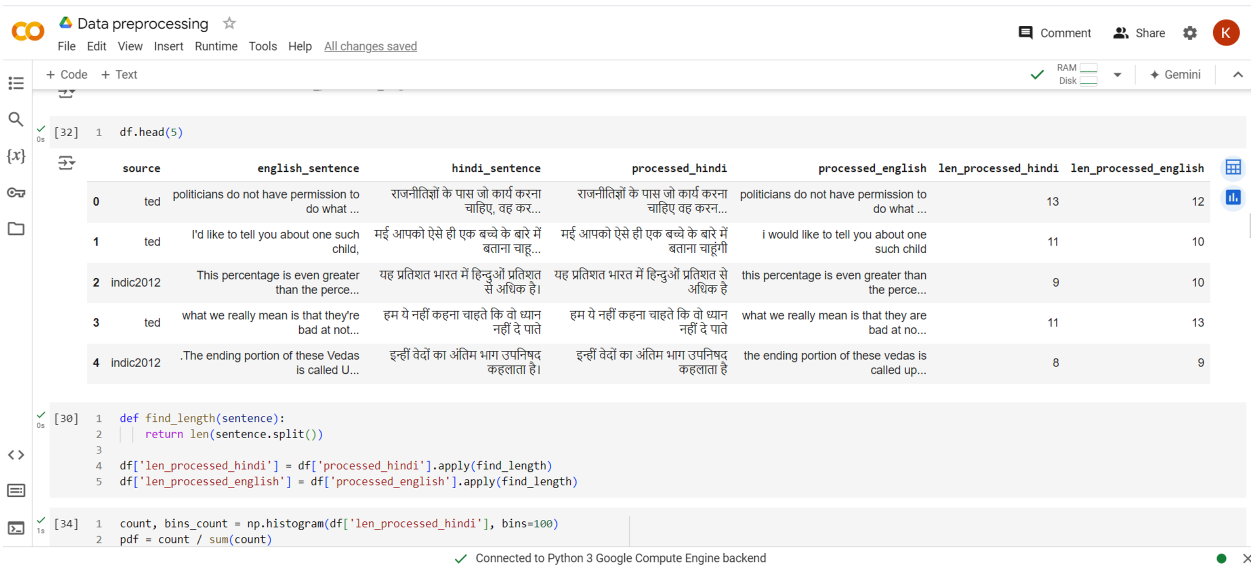
```
<Axes: xlabel='len_processed_english', ylabel='Count'>
```



```
[31]
sns.histplot(df['len_processed_english'],bins=100)
```

```
<Axes: xlabel='len_processed_english', ylabel='Count'>
```

```
[32]
df.head(5)
```



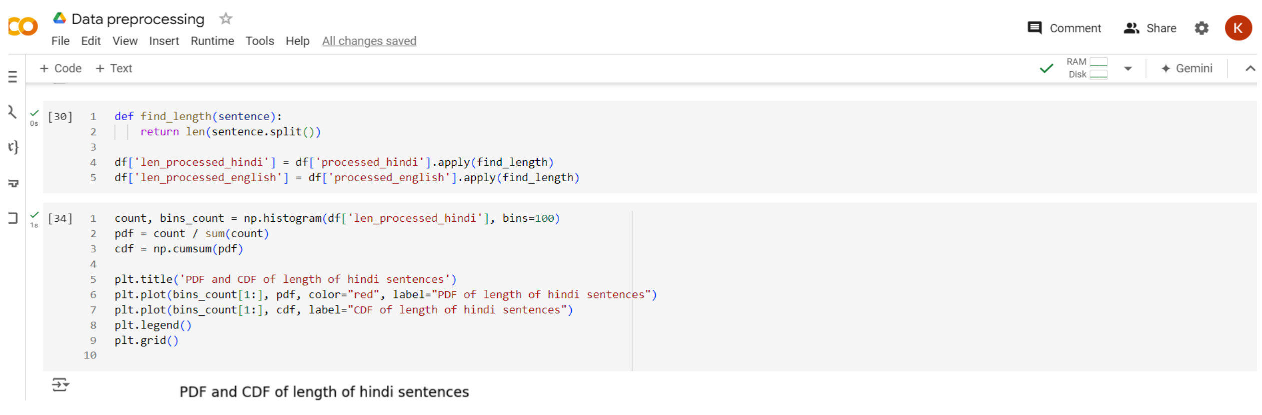
```
[32] df.head(5)
```

	source	english_sentence	hindi_sentence	processed_hindi	processed_english	len_processed_hindi	len_processed_english
0	ted	politicians do not have permission to do what ...	राजनीतिज्ञों के पास जो कार्य करना चाहिए, वह कर...	राजनीतिज्ञों के पास जो कार्य करना चाहिए वह करन...	politicians do not have permission to do what ...	13	12
1	ted	I'd like to tell you about one such child,	मई आपको ऐसे ही एक बच्चे के बारे में बताना चाहू...	मई आपको ऐसे ही एक बच्चे के बारे में बताना चाहूंगी	i would like to tell you about one such child	11	10
2	indic2012	This percentage is even greater than the perce...	यह प्रतिशत भारत में हिन्दुओं प्रतिशत से अधिक है।	यह प्रतिशत भारत में हिन्दुओं प्रतिशत से अधिक है	this percentage is even greater than the perce...	9	10
3	ted	what we really mean is that they're bad at not...	हम ये नहीं कहना चाहते कि वो ध्यान नहीं दे पाते	हम ये नहीं कहना चाहते कि वो ध्यान नहीं दे पाते	what we really mean is that they are bad at no...	11	13
4	indic2012	.The ending portion of these Vedas is called U...	इन्हीं वेदों का अंतिम भाग उपनिषद कहलाता है।	इन्हीं वेदों का अंतिम भाग उपनिषद कहलाता है	the ending portion of these vedas is called up...	8	9

```
[30] def find_length(sentence):
         return len(sentence.split())

     df['len_processed_hindi'] = df['processed_hindi'].apply(find_length)
     df['len_processed_english'] = df['processed_english'].apply(find_length)
```

```
[34] count, bins_count = np.histogram(df['len_processed_hindi'], bins=100)
     pdf = count / sum(count)
```

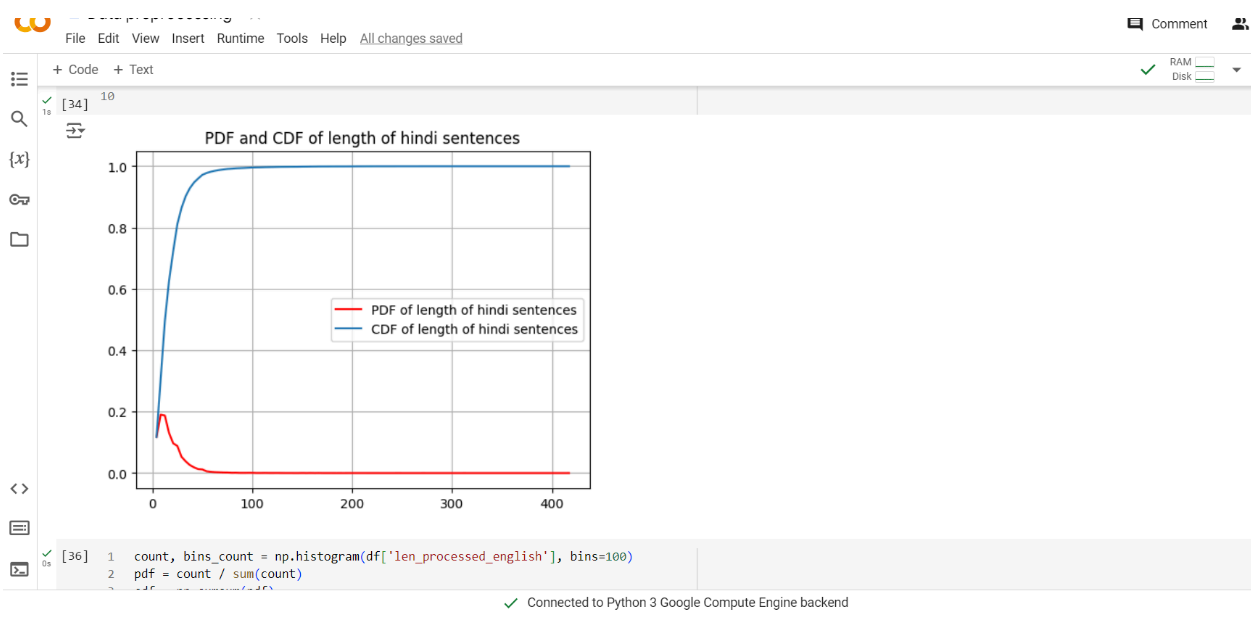
Connected to Python 3 Google Compute Engine backend



```
[30] def find_length(sentence):
         return len(sentence.split())

     df['len_processed_hindi'] = df['processed_hindi'].apply(find_length)
     df['len_processed_english'] = df['processed_english'].apply(find_length)
```

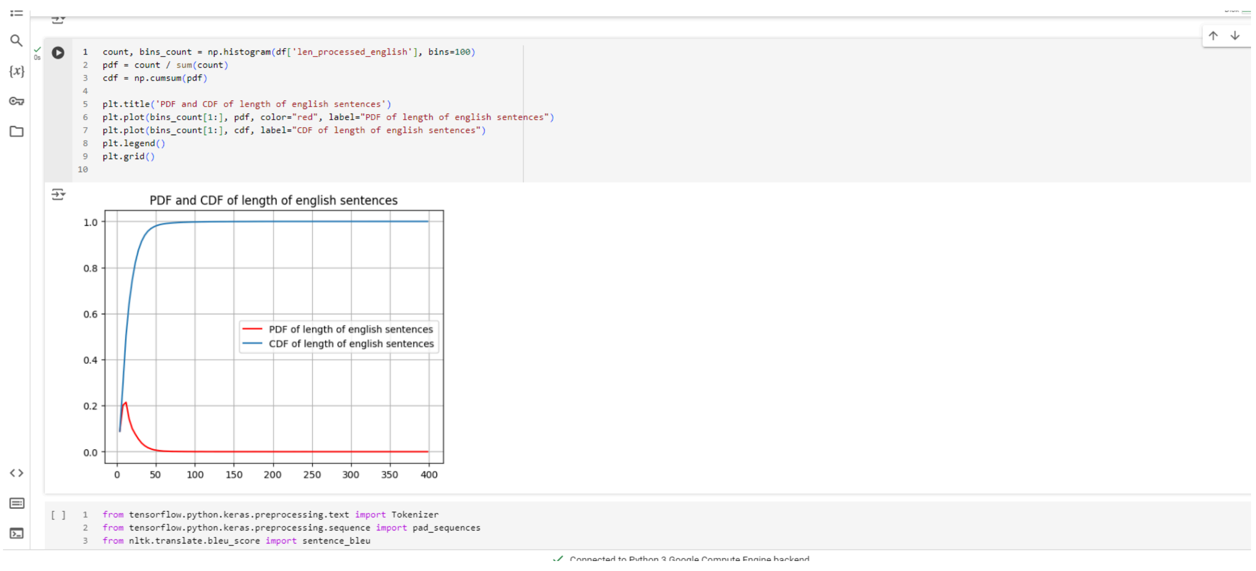
```
[34] count, bins_count = np.histogram(df['len_processed_hindi'], bins=100)
     pdf = count / sum(count)
     cdf = np.cumsum(pdf)

     plt.title('PDF and CDF of length of hindi sentences')
     plt.plot(bins_count[1:], pdf, color="red", label="PDF of length of hindi sentences")
     plt.plot(bins_count[1:], cdf, label="CDF of length of hindi sentences")
     plt.legend()
     plt.grid()
```

PDF and CDF of length of hindi sentences


File Edit View Insert Runtime Tools Help All changes saved
Comment
+ Code + Text
RAM Disk
[34]
PDF and CDF of length of hindi sentences
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
PDF of length of hindi sentences
CDF of length of hindi sentences

```
[36] count, bins_count = np.histogram(df['len_processed_english'], bins=100)
     pdf = count / sum(count)
```

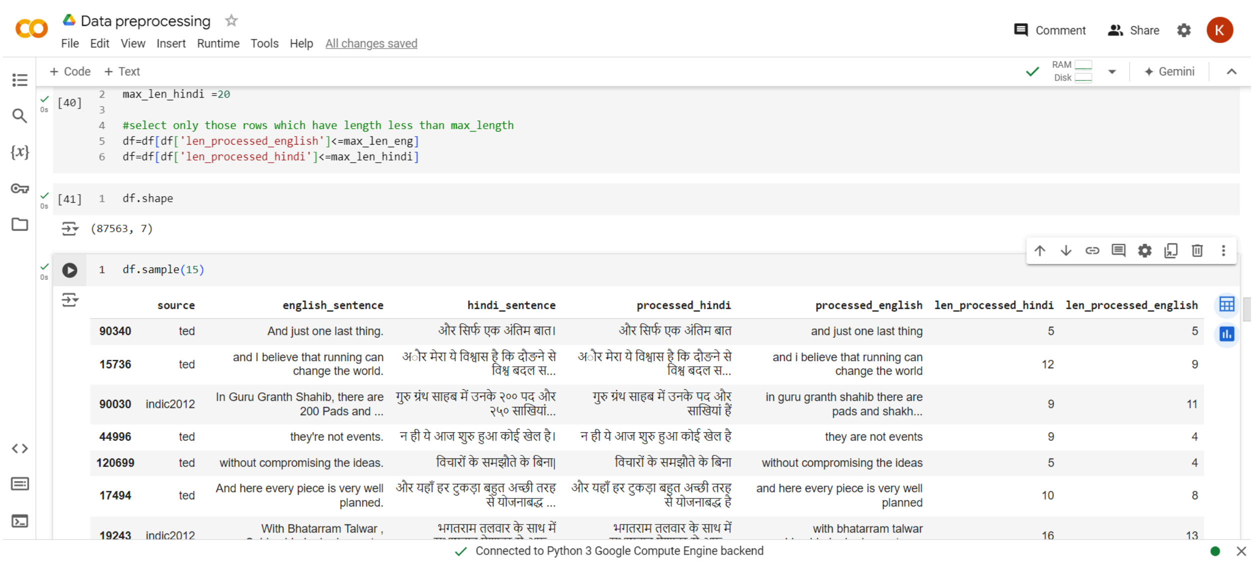
Connected to Python 3 Google Compute Engine backend



```
count, bins_count = np.histogram(df['len_processed_english'], bins=100)
pdf = count / sum(count)
cdf = np.cumsum(pdf)

plt.title('PDF and CDF of length of english sentences')
plt.plot(bins_count[1:], pdf, color="red", label="PDF of length of english sentences")
plt.plot(bins_count[1:], cdf, label="CDF of length of english sentences")
plt.legend()
plt.grid()
```

PDF and CDF of length of english sentences
PDF of length of english sentences
CDF of length of english sentences
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250
300
350
400

```
from tensorflow.python.keras.preprocessing.text import Tokenizer
from tensorflow.python.keras.preprocessing.sequence import pad_sequences
from nltk.translate.bleu_score import sentence_bleu
```



```
[40]
max_len_hindi =20

#select only those rows which have length less than max_length
df=df[df['len_processed_english']<=max_len_eng]
df=df[df['len_processed_hindi']<=max_len_hindi]
```

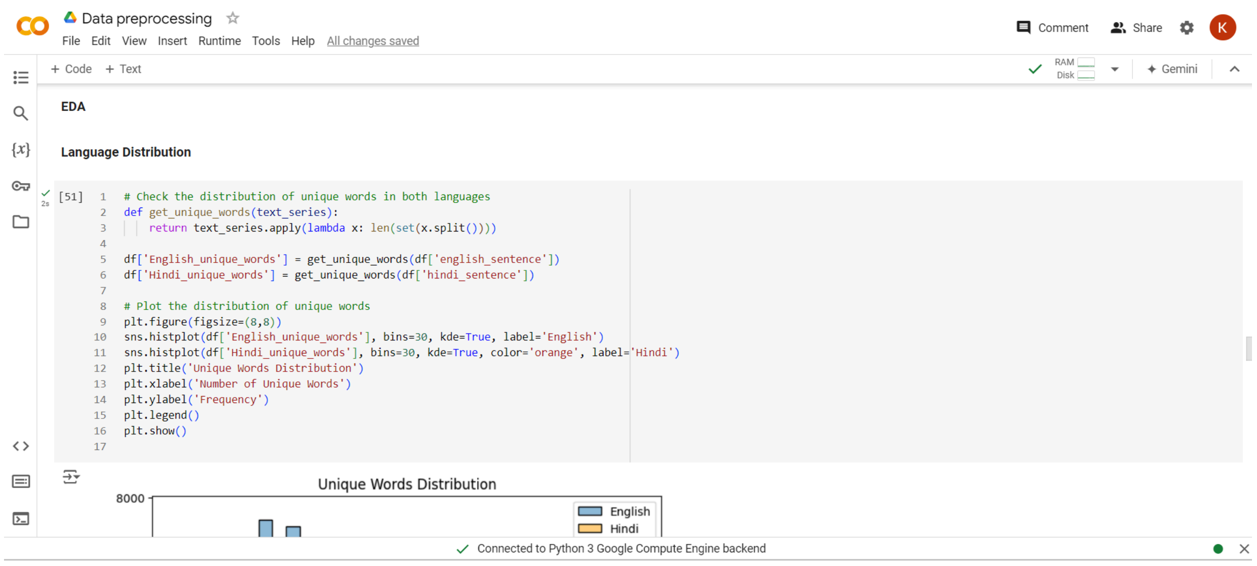
```
[41]
df.shape
```

```
(87563, 7)
```

```
df.sample(15)
```

	source	english_sentence	hindi_sentence	processed_hindi	processed_english	len_processed_hindi	len_processed_english
90340	ted	And just one last thing.	और सिर्फ एक अंतिम बात।	और सिर्फ एक अंतिम बात	and just one last thing	5	5
15736	ted	and I believe that running can change the world.	अ ोर मेरा ये विश्वास है कि दौड्ने से विश्व बदल स...	अ ोर मेरा ये विश्वास है कि दौड्ने से विश्व बदल स...	and i believe that running can change the world	12	9
90030	indic2012	In Guru Granth Shahib, there are 200 Pads and ...	गुरु ग्रंथ साहब में उनके २०० पद और २५० साखियां...	गुरु ग्रंथ साहब में उनके पद और साखियां हैं	in guru granth shahib there are pads and shakh...	9	11
44996	ted	they're not events.	न ही ये आज शुरु हुआ कोई खेल है।	न ही ये आज शुरु हुआ कोई खेल है	they are not events	9	4
120699	ted	without compromising the ideas.	विचारों के समझौते के बिना|	विचारों के समझौते के बिना	without compromising the ideas	5	4
17494	ted	And here every piece is very well planned.	और यहाँ हर टुकड़ा बहुत अच्छी तरह से योजनाबद्ध ...	और यहाँ हर टुकड़ा बहुत अच्छी तरह से योजनाबद्ध है	and here every piece is very well planned	10	8
19243	indic2012	With Bhatarram Talwar ,	भगतराम तलवार के साथ में	भगतराम तलवार के साथ में	with bhatarram talwar	16	13

EDA(Exploratory Data Analysis)

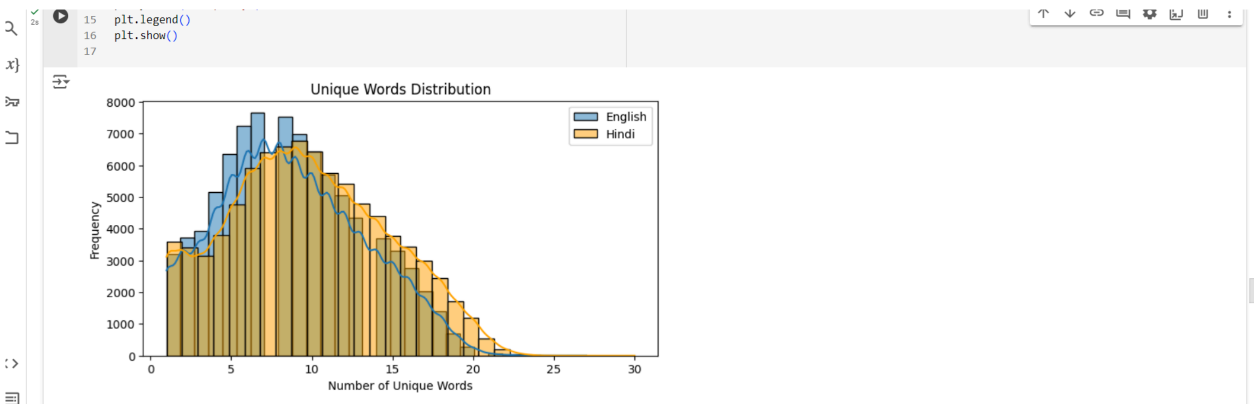


EDA

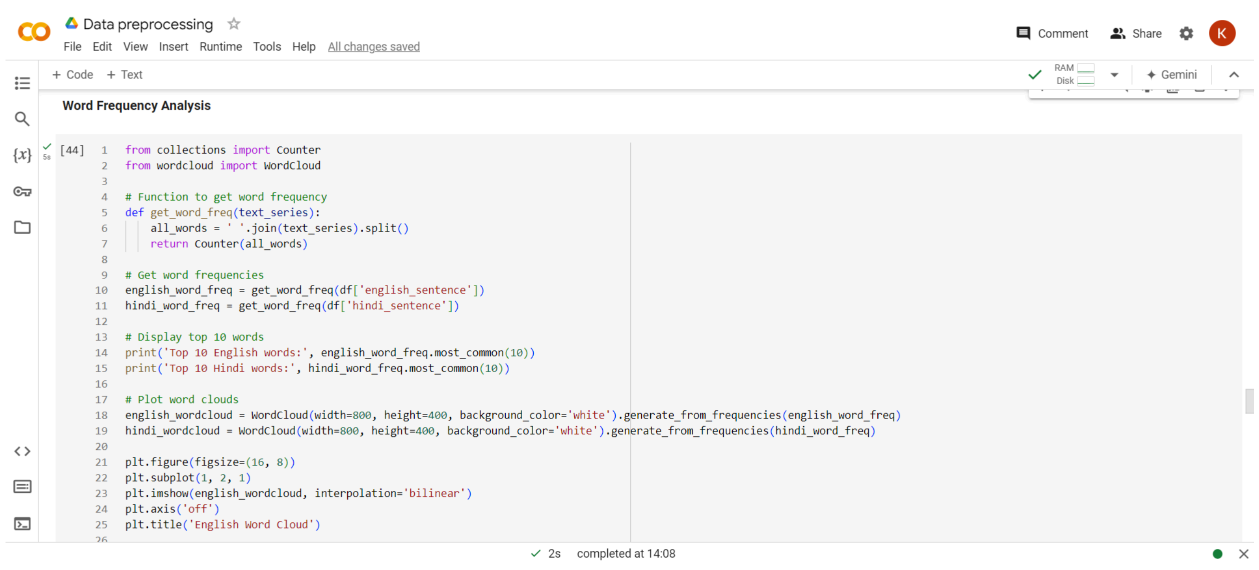
Language Distribution

```
[51]
# Check the distribution of unique words in both languages
def get_unique_words(text_series):
    return text_series.apply(lambda x: len(set(x.split())))

df['English_unique_words'] = get_unique_words(df['english_sentence'])
df['Hindi_unique_words'] = get_unique_words(df['hindi_sentence'])

# Plot the distribution of unique words
plt.figure(figsize=(8,8))
sns.histplot(df['English_unique_words'], bins=30, kde=True, label='English')
sns.histplot(df['Hindi_unique_words'], bins=30, kde=True, color='orange', label='Hindi')
plt.title('Unique Words Distribution')
plt.xlabel('Number of Unique Words')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

Unique Words Distribution
8000
English
Hindi
Connected to Python 3 Google Compute Engine backend



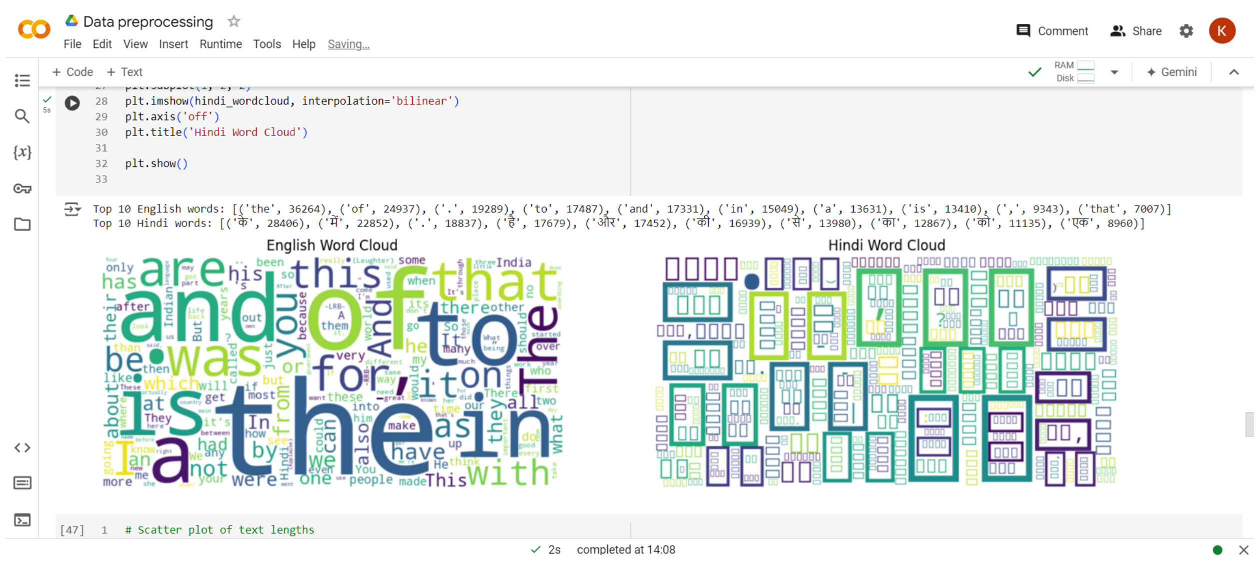
```
plt.legend()
plt.show()
```



Word Frequency Analysis

```
from collections import Counter
from wordcloud import WordCloud

# Function to get word frequency
def get_word_freq(text_series):
    all_words = ' '.join(text_series).split()
    return Counter(all_words)

# Get word frequencies
english_word_freq = get_word_freq(df['english_sentence'])
hindi_word_freq = get_word_freq(df['hindi_sentence'])

# Display top 10 words
print('Top 10 English words:', english_word_freq.most_common(10))
print('Top 10 Hindi words:', hindi_word_freq.most_common(10))

# Plot word clouds
english_wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(english_word_freq)
hindi_wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(hindi_word_freq)

plt.figure(figsize=(16, 8))
plt.subplot(1, 2, 1)
plt.imshow(english_wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('English Word Cloud')
```

✓ 2s completed at 14:08

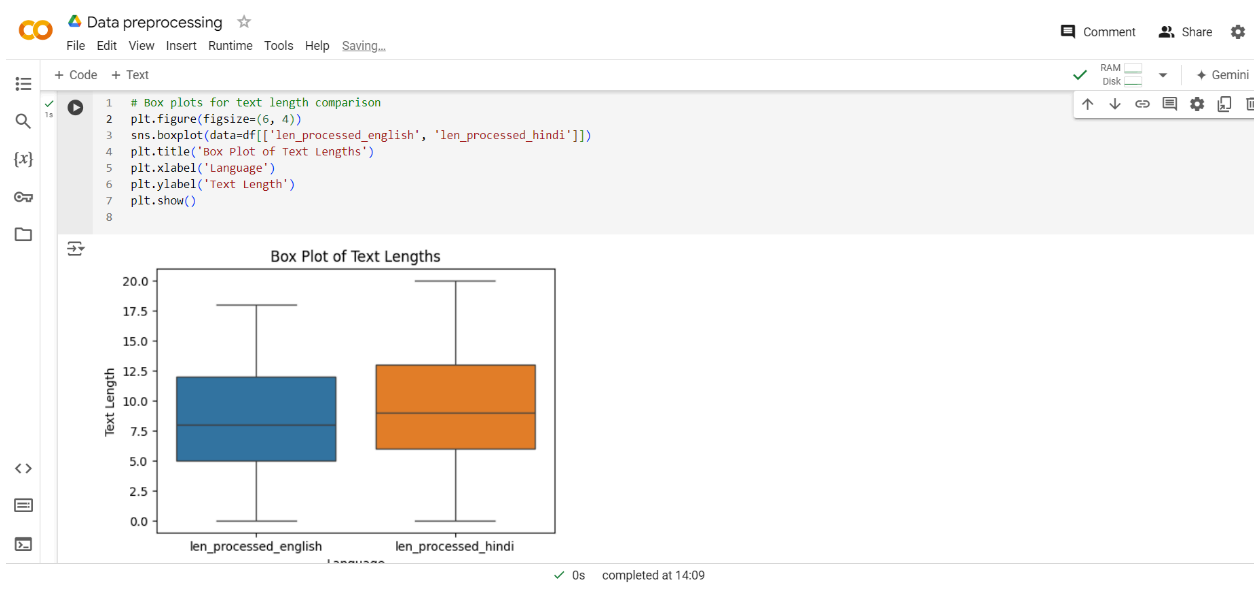


```
plt.imshow(hindi_wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Hindi Word Cloud')

plt.show()
```

```
Top 10 English words: [('the', 36264), ('of', 24937), ('.', 19289), ('to', 17487), ('and', 17331), ('in', 15049), ('a', 13631), ('is', 13410), (',', 9343), ('that', 7007)]
Top 10 Hindi words: [('के', 28406), ('में', 22852), ('.', 18837), ('है', 17679), ('और', 17452), ('की', 16939), ('से', 13980), ('का', 12867), ('को', 11135), ('एक', 8960)]
```

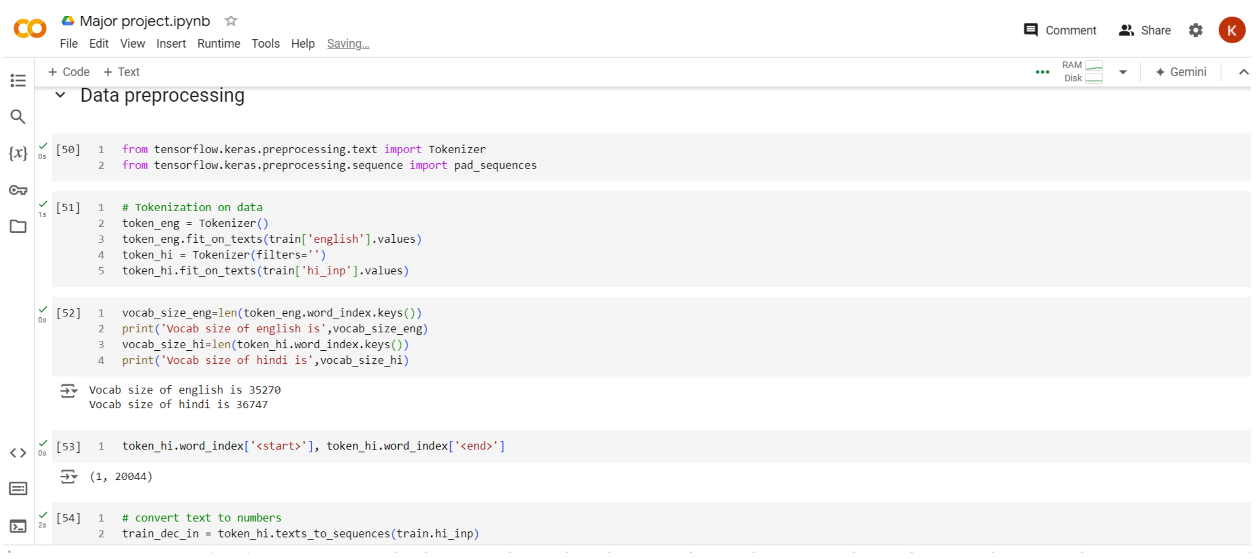
```
# Scatter plot of text lengths
```

✓ 2s completed at 14:08



```
# Box plots for text length comparison
plt.figure(figsize=(6, 4))
sns.boxplot(data=df[['len_processed_english', 'len_processed_hindi']])
plt.title('Box Plot of Text Lengths')
plt.xlabel('Language')
plt.ylabel('Text Length')
plt.show()
```

✓ 0s completed at 14:09



Data preprocessing

```
[50]
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
[51]
# Tokenization on data
token_eng = Tokenizer()
token_eng.fit_on_texts(train['english'].values)
token_hi = Tokenizer(filters='')
token_hi.fit_on_texts(train['hi_inp'].values)
```

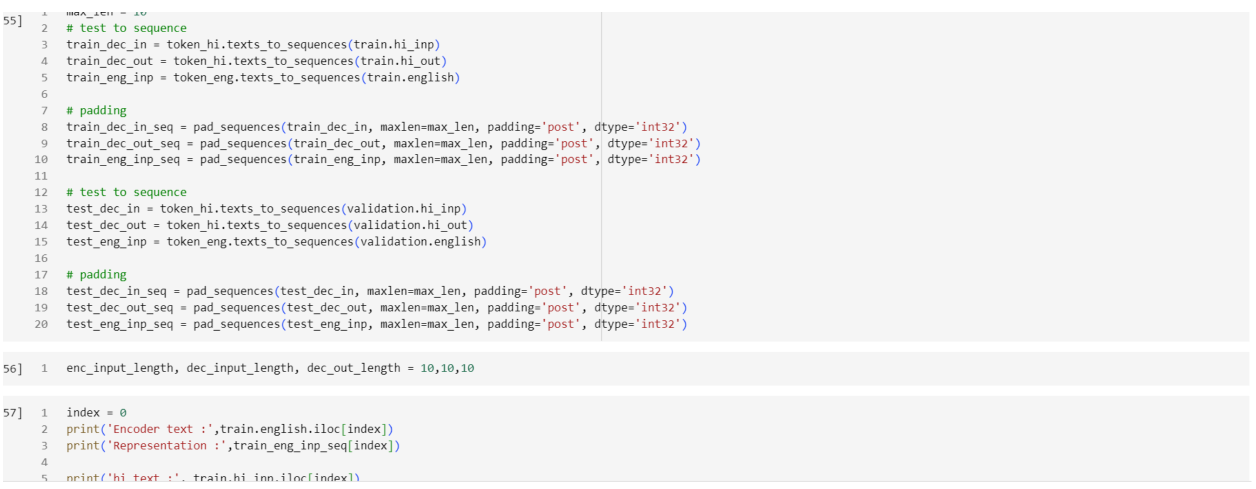
```
[52]
vocab_size_eng=len(token_eng.word_index.keys())
print('Vocab size of english is',vocab_size_eng)
vocab_size_hi=len(token_hi.word_index.keys())
print('Vocab size of hindi is',vocab_size_hi)
```

```
Vocab size of english is 35270
Vocab size of hindi is 36747
```

```
[53]
token_hi.word_index['<start>'], token_hi.word_index['<end>']
```

```
(1, 20044)
```

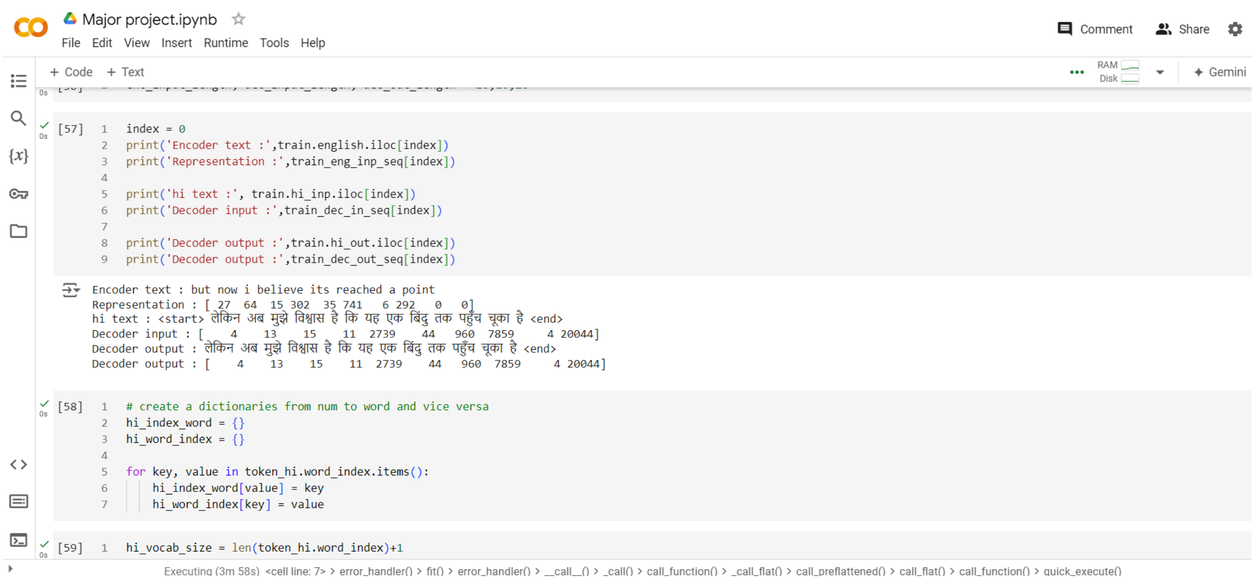
```
[54]
# convert text to numbers
train_dec_in = token_hi.texts_to_sequences(train.hi_inp)
```



```
55]
max_len = 10
# test to sequence
train_dec_in = token_hi.texts_to_sequences(train.hi_inp)
train_dec_out = token_hi.texts_to_sequences(train.hi_out)
train_eng_inp = token_eng.texts_to_sequences(train.english)

# padding
train_dec_in_seq = pad_sequences(train_dec_in, maxlen=max_len, padding='post', dtype='int32')
train_dec_out_seq = pad_sequences(train_dec_out, maxlen=max_len, padding='post', dtype='int32')
train_eng_inp_seq = pad_sequences(train_eng_inp, maxlen=max_len, padding='post', dtype='int32')

# test to sequence
test_dec_in = token_hi.texts_to_sequences(validation.hi_inp)
test_dec_out = token_hi.texts_to_sequences(validation.hi_out)
test_eng_inp = token_eng.texts_to_sequences(validation.english)

# padding
test_dec_in_seq = pad_sequences(test_dec_in, maxlen=max_len, padding='post', dtype='int32')
test_dec_out_seq = pad_sequences(test_dec_out, maxlen=max_len, padding='post', dtype='int32')
test_eng_inp_seq = pad_sequences(test_eng_inp, maxlen=max_len, padding='post', dtype='int32')
```

```
56]
enc_input_length, dec_input_length, dec_out_length = 10,10,10
```

```
57]
index = 0
print('Encoder text :',train.english.iloc[index])
print('Representation :',train_eng_inp_seq[index])

print('hi text :', train.hi_inp.iloc[index])
```



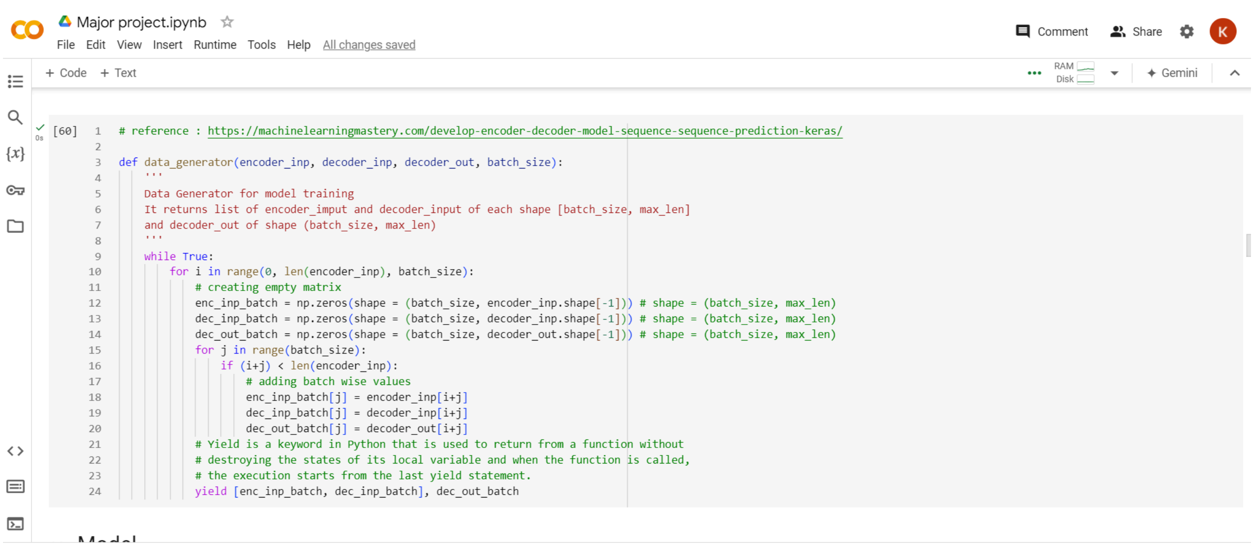
```
[57]
index = 0
print('Encoder text :',train.english.iloc[index])
print('Representation :',train_eng_inp_seq[index])

print('hi text :', train.hi_inp.iloc[index])
print('Decoder input :',train_dec_in_seq[index])

print('Decoder output :',train.hi_out.iloc[index])
print('Decoder output :',train_dec_out_seq[index])
```

```
Encoder text : but now i believe its reached a point
Representation : [ 27  64  15 302  35 741   6 292   0   0]
hi text : <start> लेकिन अब मुझे विश्वास है कि यह एक बिंदु तक पहुँच चूका है <end>
Decoder input : [    4    13    15    11  2739    44   960  7859     4 20044]
Decoder output : लेकिन अब मुझे विश्वास है कि यह एक बिंदु तक पहुँच चूका है <end>
Decoder output : [    4    13    15    11  2739    44   960  7859     4 20044]
```

```
[58]
# create a dictionaries from num to word and vice versa
hi_index_word = {}
hi_word_index = {}

for key, value in token_hi.word_index.items():
    hi_index_word[value] = key
    hi_word_index[key] = value
```

```
[59]
hi_vocab_size = len(token_hi.word_index)+1
```

```
Executing (3m 58s) <cell line: 7> > error_handler() > fit() > error_handler() > __call__() > _call() > call_function() > _call_flat() > call_preflattened() > call_flat() > call_function() > quick_execute()
```

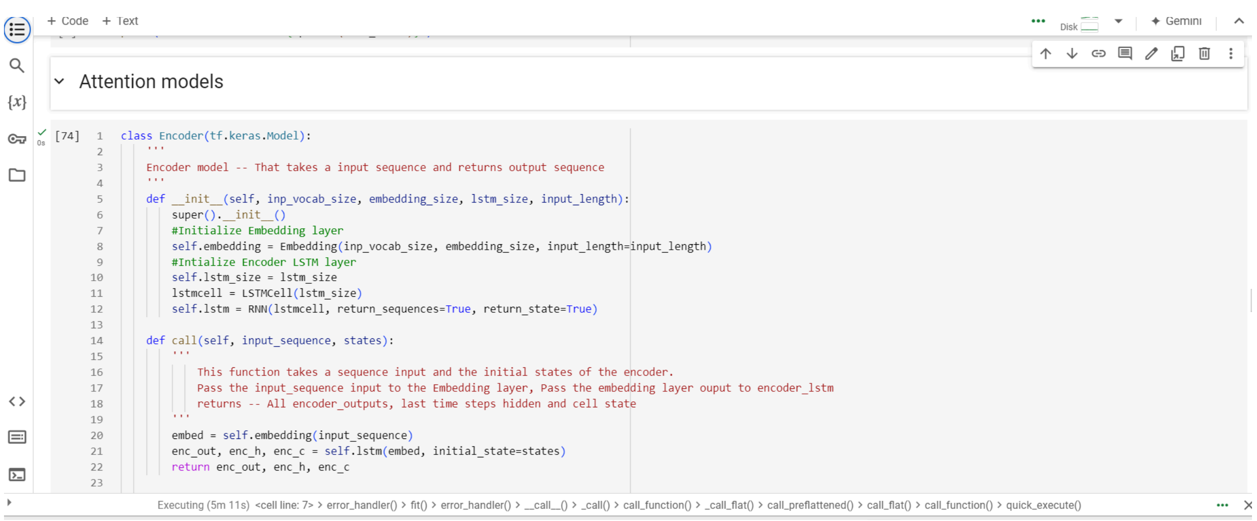


Major project.ipynb

File Edit View Insert Runtime Tools Help All changes saved

[60]

```python
# reference : https://machinelearningmastery.com/develop-encoder-decoder-model-sequence-sequence-prediction-keras/

def data_generator(encoder_inp, decoder_inp, decoder_out, batch_size):
    '''
    Data Generator for model training
    It returns list of encoder_imput and decoder_input of each shape [batch_size, max_len]
    and decoder_out of shape (batch_size, max_len)
    '''
    while True:
        for i in range(0, len(encoder_inp), batch_size):
            # creating empty matrix
            enc_inp_batch = np.zeros(shape = (batch_size, encoder_inp.shape[-1])) # shape = (batch_size, max_len)
            dec_inp_batch = np.zeros(shape = (batch_size, decoder_inp.shape[-1])) # shape = (batch_size, max_len)
            dec_out_batch = np.zeros(shape = (batch_size, decoder_out.shape[-1])) # shape = (batch_size, max_len)
            for j in range(batch_size):
                if (i+j) < len(encoder_inp):
                    # adding batch wise values
                    enc_inp_batch[j] = encoder_inp[i+j]
                    dec_inp_batch[j] = decoder_inp[i+j]
                    dec_out_batch[j] = decoder_out[i+j]
            # Yield is a keyword in Python that is used to return from a function without
            # destroying the states of its local variable and when the function is called,
            # the execution starts from the last yield statement.
            yield [enc_inp_batch, dec_inp_batch], dec_out_batch
```

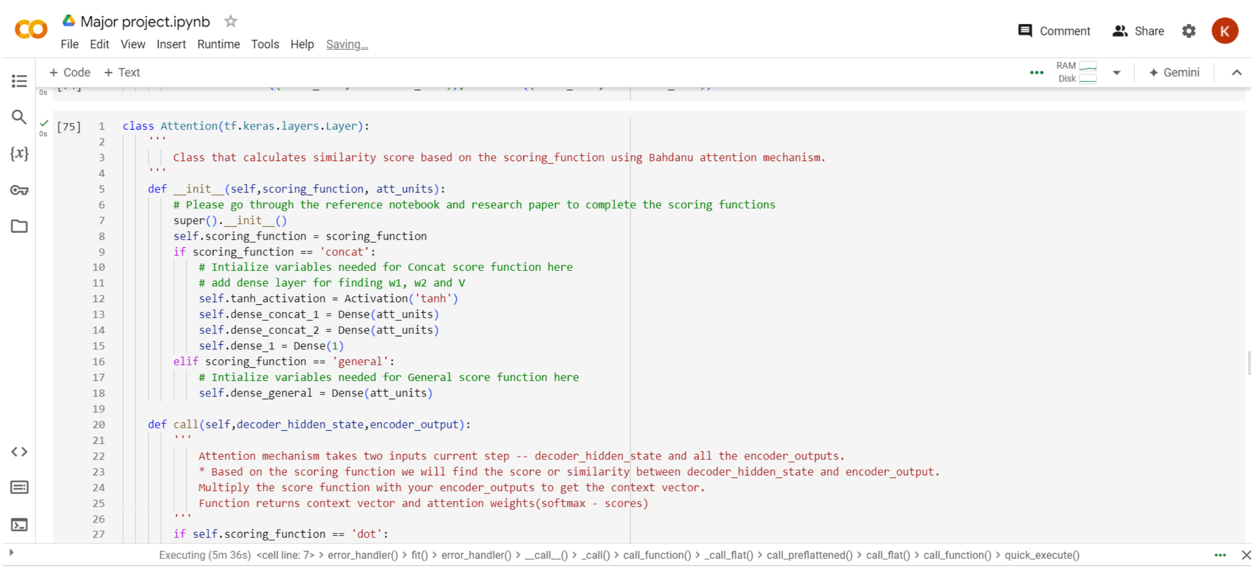


Attention models

[74]

```python
class Encoder(tf.keras.Model):
    '''
    Encoder model -- That takes a input sequence and returns output sequence
    '''
    def __init__(self, inp_vocab_size, embedding_size, lstm_size, input_length):
        super().__init__()
        #Initialize Embedding layer
        self.embedding = Embedding(inp_vocab_size, embedding_size, input_length=input_length)
        #Intialize Encoder LSTM layer
        self.lstm_size = lstm_size
        lstmcell = LSTMCell(lstm_size)
        self.lstm = RNN(lstmcell, return_sequences=True, return_state=True)

    def call(self, input_sequence, states):
        '''
          This function takes a sequence input and the initial states of the encoder.
          Pass the input_sequence input to the Embedding layer, Pass the embedding layer ouput to encoder_lstm
          returns -- All encoder_outputs, last time steps hidden and cell state
        '''
        embed = self.embedding(input_sequence)
        enc_out, enc_h, enc_c = self.lstm(embed, initial_state=states)
        return enc_out, enc_h, enc_c

```

Executing (5m 11s) <cell line: 7> › error_handler() › fit() › error_handler() › __call__() › _call() › call_function() › _call_flat() › call_preflattened() › call_flat() › call_function() › quick_execute()



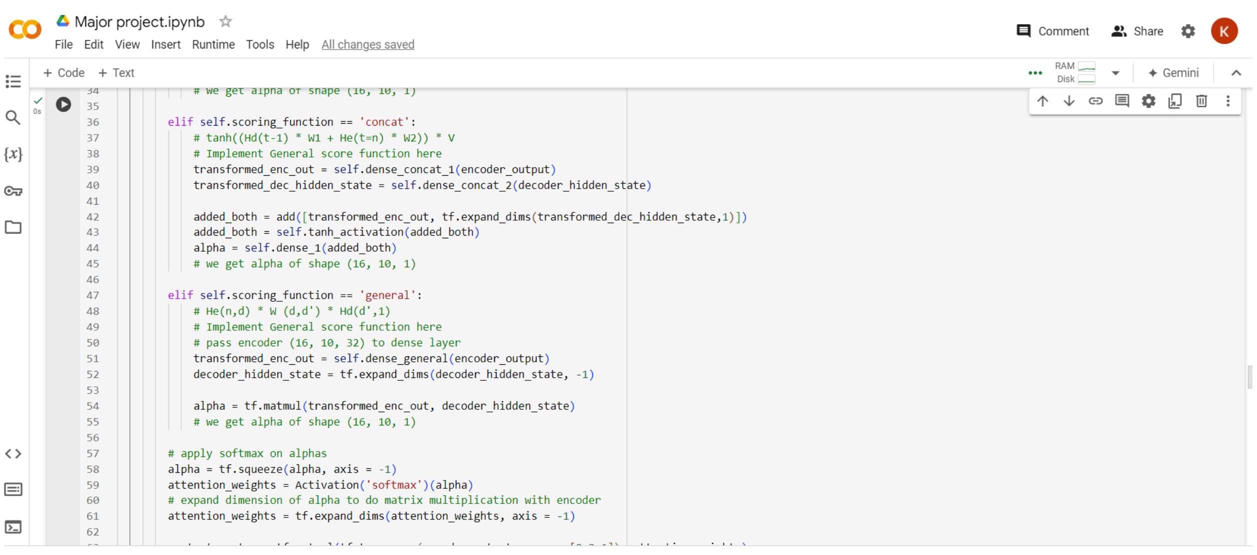
Major project.ipynb

File Edit View Insert Runtime Tools Help Saving...

[75]

```python
class Attention(tf.keras.layers.Layer):
    '''
      Class that calculates similarity score based on the scoring_function using Bahdanu attention mechanism.
    '''
    def __init__(self,scoring_function, att_units):
        # Please go through the reference notebook and research paper to complete the scoring functions
        super().__init__()
        self.scoring_function = scoring_function
        if scoring_function == 'concat':
            # Intialize variables needed for Concat score function here
            # add dense layer for finding w1, w2 and V
            self.tanh_activation = Activation('tanh')
            self.dense_concat_1 = Dense(att_units)
            self.dense_concat_2 = Dense(att_units)
            self.dense_1 = Dense(1)
        elif scoring_function == 'general':
            # Intialize variables needed for General score function here
            self.dense_general = Dense(att_units)

    def call(self,decoder_hidden_state,encoder_output):
        '''
          Attention mechanism takes two inputs current step -- decoder_hidden_state and all the encoder_outputs.
          * Based on the scoring function we will find the score or similarity between decoder_hidden_state and encoder_output.
          Multiply the score function with your encoder_outputs to get the context vector.
          Function returns context vector and attention weights(softmax - scores)
        '''
        if self.scoring_function == 'dot':
```

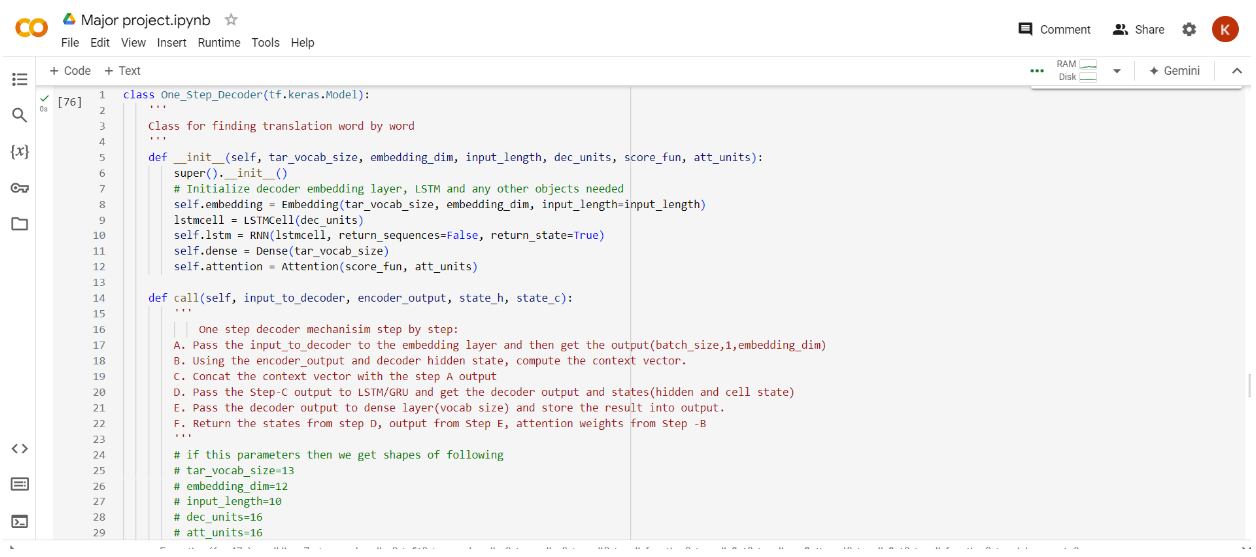
Executing (5m 36s) <cell line: 7> › error_handler() › fit() › error_handler() › __call__() › _call() › call_function() › _call_flat() › call_preflattened() › call_flat() › call_function() › quick_execute()



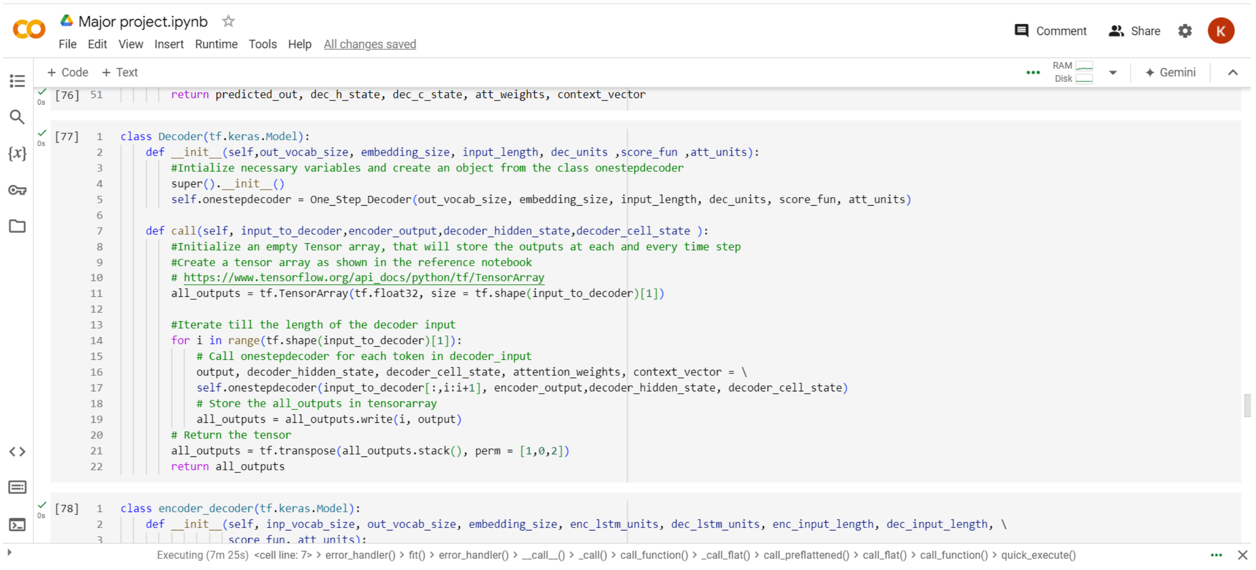
Major project.ipynb

File Edit View Insert Runtime Tools Help All changes saved

```python
            # we get alpha of shape (16, 10, 1)

        elif self.scoring_function == 'concat':
            # tanh((Hd(t-1) * W1 + He(t=n) * W2)) * V
            # Implement General score function here
            transformed_enc_out = self.dense_concat_1(encoder_output)
            transformed_dec_hidden_state = self.dense_concat_2(decoder_hidden_state)

            added_both = add([transformed_enc_out, tf.expand_dims(transformed_dec_hidden_state,1)])
            added_both = self.tanh_activation(added_both)
            alpha = self.dense_1(added_both)
            # we get alpha of shape (16, 10, 1)

        elif self.scoring_function == 'general':
            # He(n,d) * W (d,d') * Hd(d',1)
            # Implement General score function here
            # pass encoder (16, 10, 32) to dense layer
            transformed_enc_out = self.dense_general(encoder_output)
            decoder_hidden_state = tf.expand_dims(decoder_hidden_state, -1)

            alpha = tf.matmul(transformed_enc_out, decoder_hidden_state)
            # we get alpha of shape (16, 10, 1)

        # apply softmax on alphas
        alpha = tf.squeeze(alpha, axis = -1)
        attention_weights = Activation('softmax')(alpha)
        # expand dimension of alpha to do matrix multiplication with encoder
        attention_weights = tf.expand_dims(attention_weights, axis = -1)

```

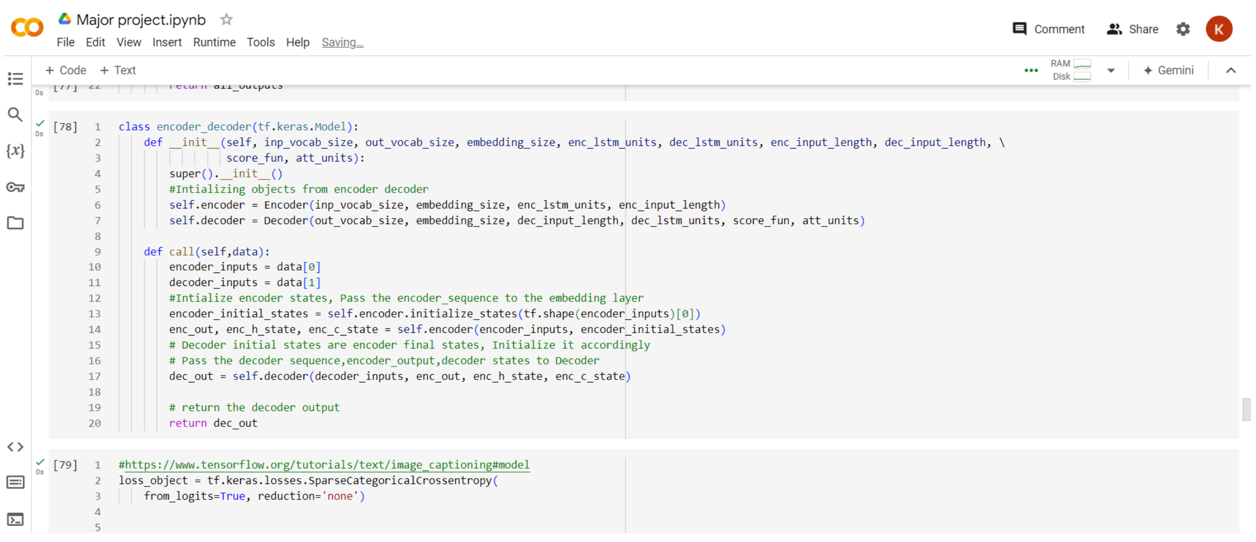


```
[76]
class One_Step_Decoder(tf.keras.Model):
    '''
    Class for finding translation word by word
    '''
    def __init__(self, tar_vocab_size, embedding_dim, input_length, dec_units, score_fun, att_units):
        super().__init__()
        # Initialize decoder embedding layer, LSTM and any other objects needed
        self.embedding = Embedding(tar_vocab_size, embedding_dim, input_length=input_length)
        lstmcell = LSTMCell(dec_units)
        self.lstm = RNN(lstmcell, return_sequences=False, return_state=True)
        self.dense = Dense(tar_vocab_size)
        self.attention = Attention(score_fun, att_units)

    def call(self, input_to_decoder, encoder_output, state_h, state_c):
        '''
            One step decoder mechanisim step by step:
        A. Pass the input_to_decoder to the embedding layer and then get the output(batch_size,1,embedding_dim)
        B. Using the encoder_output and decoder hidden state, compute the context vector.
        C. Concat the context vector with the step A output
        D. Pass the Step-C output to LSTM/GRU and get the decoder output and states(hidden and cell state)
        E. Pass the decoder output to dense layer(vocab size) and store the result into output.
        F. Return the states from step D, output from Step E, attention weights from Step -B
        '''
        # if this parameters then we get shapes of following
        # tar_vocab_size=13
        # embedding_dim=12
        # input_length=10
        # dec_units=16
        # att_units=16
```

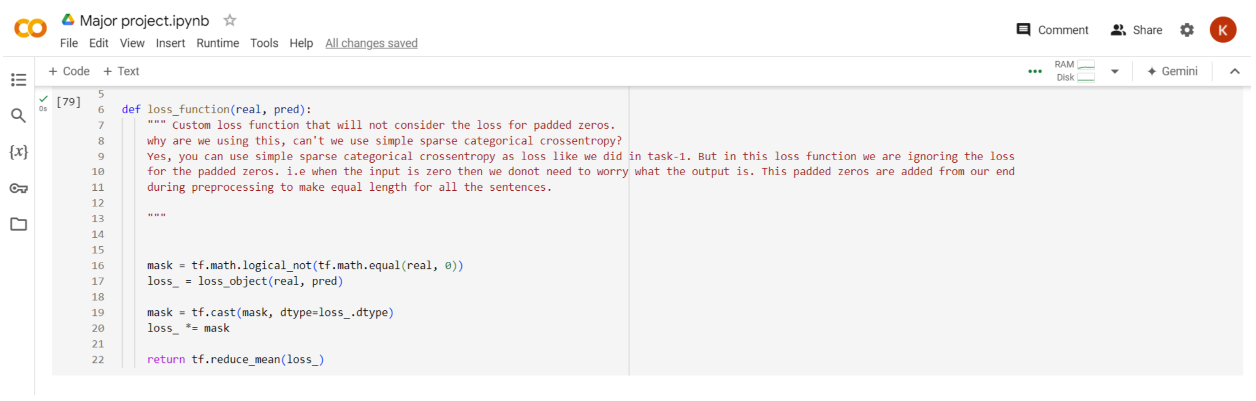


```
[76] 51          return predicted_out, dec_h_state, dec_c_state, att_weights, context_vector

[77]
class Decoder(tf.keras.Model):
    def __init__(self,out_vocab_size, embedding_size, input_length, dec_units ,score_fun ,att_units):
        #Intialize necessary variables and create an object from the class onestepdecoder
        super().__init__()
        self.onestepdecoder = One_Step_Decoder(out_vocab_size, embedding_size, input_length, dec_units, score_fun, att_units)

    def call(self, input_to_decoder,encoder_output,decoder_hidden_state,decoder_cell_state ):
        #Initialize an empty Tensor array, that will store the outputs at each and every time step
        #Create a tensor array as shown in the reference notebook
        # https://www.tensorflow.org/api_docs/python/tf/TensorArray
        all_outputs = tf.TensorArray(tf.float32, size = tf.shape(input_to_decoder)[1])

        #Iterate till the length of the decoder input
        for i in range(tf.shape(input_to_decoder)[1]):
            # Call onestepdecoder for each token in decoder_input
            output, decoder_hidden_state, decoder_cell_state, attention_weights, context_vector = \
            self.onestepdecoder(input_to_decoder[:,i:i+1], encoder_output,decoder_hidden_state, decoder_cell_state)
            # Store the all_outputs in tensorarray
            all_outputs = all_outputs.write(i, output)
        # Return the tensor
        all_outputs = tf.transpose(all_outputs.stack(), perm = [1,0,2])
        return all_outputs

[78]
class encoder_decoder(tf.keras.Model):
    def __init__(self, inp_vocab_size, out_vocab_size, embedding_size, enc_lstm_units, dec_lstm_units, enc_input_length, dec_input_length, \
                    score_fun, att_units):
```

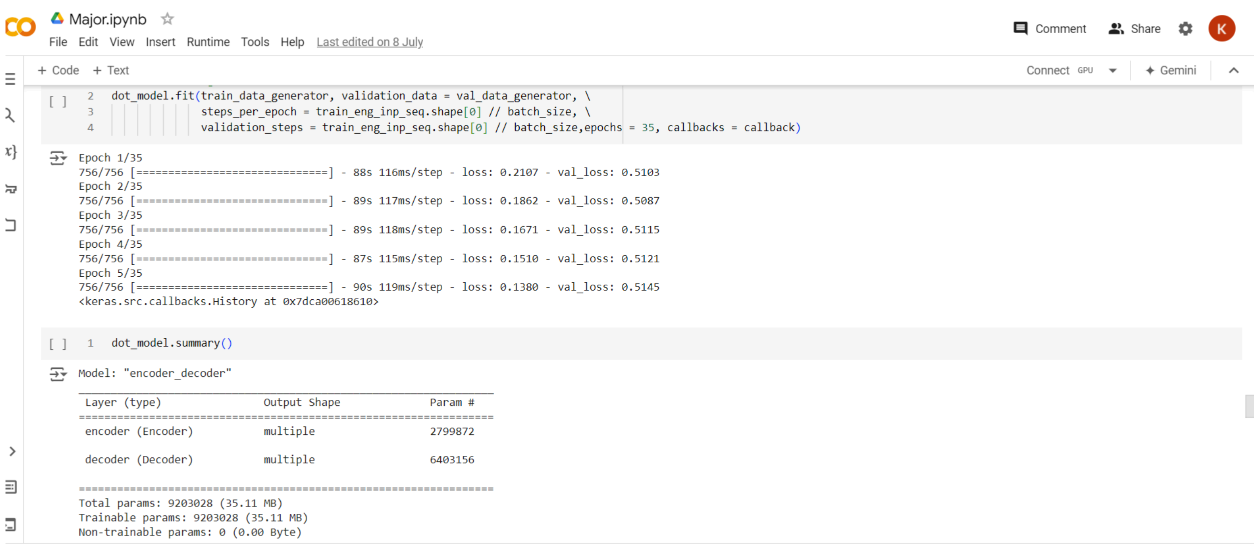
Executing (7m 25s) <cell line: 7> > error_handler() > fit() > error_handler() > __call__() > _call() > call_function() > _call_flat() > call_preflattened() > call_flat() > call_function() > quick_execute()



```
[78]
class encoder_decoder(tf.keras.Model):
    def __init__(self, inp_vocab_size, out_vocab_size, embedding_size, enc_lstm_units, dec_lstm_units, enc_input_length, dec_input_length, \
                    score_fun, att_units):
        super().__init__()
        #Intializing objects from encoder decoder
        self.encoder = Encoder(inp_vocab_size, embedding_size, enc_lstm_units, enc_input_length)
        self.decoder = Decoder(out_vocab_size, embedding_size, dec_input_length, dec_lstm_units, score_fun, att_units)

    def call(self,data):
        encoder_inputs = data[0]
        decoder_inputs = data[1]
        #Intialize encoder states, Pass the encoder_sequence to the embedding layer
        encoder_initial_states = self.encoder.initialize_states(tf.shape(encoder_inputs)[0])
        enc_out, enc_h_state, enc_c_state = self.encoder(encoder_inputs, encoder_initial_states)
        # Decoder initial states are encoder final states, Initialize it accordingly
        # Pass the decoder sequence,encoder_output,decoder states to Decoder
        dec_out = self.decoder(decoder_inputs, enc_out, enc_h_state, enc_c_state)

        # return the decoder output
        return dec_out

[79]
#https://www.tensorflow.org/tutorials/text/image_captioning#model
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')
```



```
[79]
def loss_function(real, pred):
    """ Custom loss function that will not consider the loss for padded zeros.
    why are we using this, can't we use simple sparse categorical crossentropy?
    Yes, you can use simple sparse categorical crossentropy as loss like we did in task-1. But in this loss function we are ignoring the loss
    for the padded zeros. i.e when the input is zero then we donot need to worry what the output is. This padded zeros are added from our end
    during preprocessing to make equal length for all the sentences.

    """


    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)
```

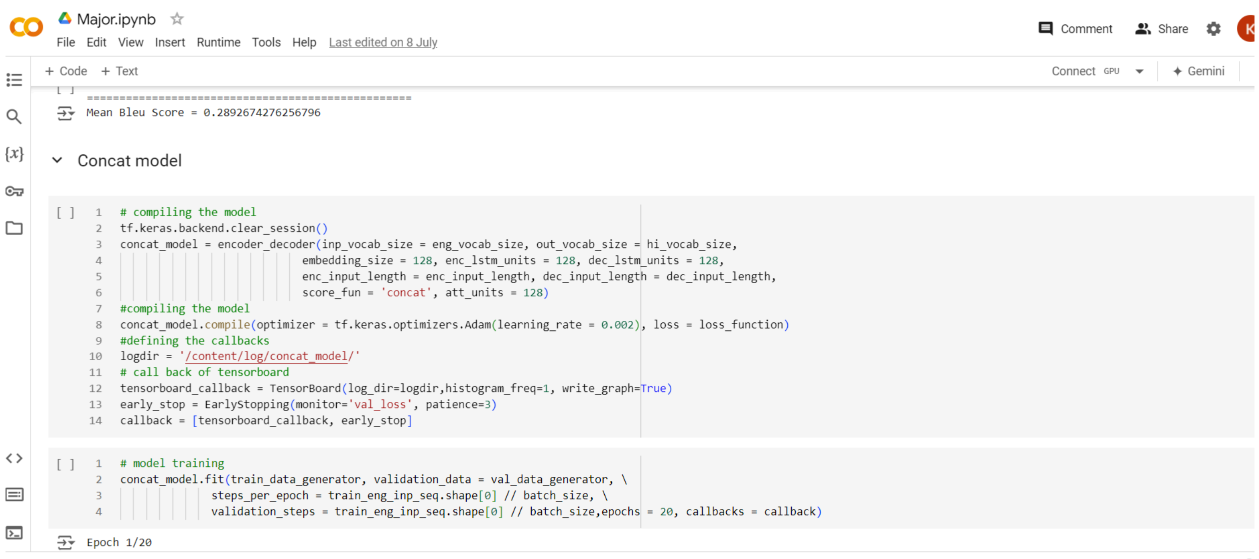


```
dot_model.fit(train_data_generator, validation_data = val_data_generator, \
              steps_per_epoch = train_eng_inp_seq.shape[0] // batch_size, \
              validation_steps = train_eng_inp_seq.shape[0] // batch_size,epochs = 35, callbacks = callback)
```

```
Epoch 1/35
756/756 [==============================] - 88s 116ms/step - loss: 0.2107 - val_loss: 0.5103
Epoch 2/35
756/756 [==============================] - 89s 117ms/step - loss: 0.1862 - val_loss: 0.5087
Epoch 3/35
756/756 [==============================] - 89s 118ms/step - loss: 0.1671 - val_loss: 0.5115
Epoch 4/35
756/756 [==============================] - 87s 115ms/step - loss: 0.1510 - val_loss: 0.5121
Epoch 5/35
756/756 [==============================] - 90s 119ms/step - loss: 0.1380 - val_loss: 0.5145
<keras.src.callbacks.History at 0x7dca00618610>
```

```
dot_model.summary()
```

```
Model: "encoder_decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder (Encoder)           multiple                  2799872

 decoder (Decoder)           multiple                  6403156

=================================================================
Total params: 9203028 (35.11 MB)
Trainable params: 9203028 (35.11 MB)
Non-trainable params: 0 (0.00 Byte)
```



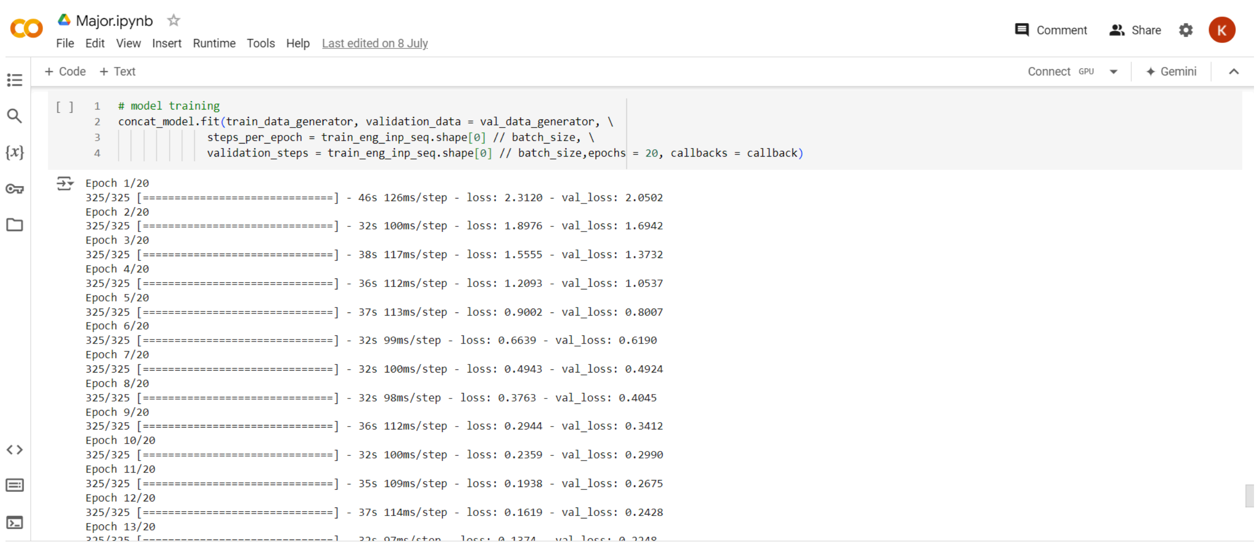
```
==================================================
Mean Bleu Score = 0.2892674276256796
```

Concat model

```
# compiling the model
tf.keras.backend.clear_session()
concat_model = encoder_decoder(inp_vocab_size = eng_vocab_size, out_vocab_size = hi_vocab_size,
                               embedding_size = 128, enc_lstm_units = 128, dec_lstm_units = 128,
                               enc_input_length = enc_input_length, dec_input_length = dec_input_length,
                               score_fun = 'concat', att_units = 128)
#compiling the model
concat_model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.002), loss = loss_function)
#defining the callbacks
logdir = '/content/log/concat_model/'
# call back of tensorboard
tensorboard_callback = TensorBoard(log_dir=logdir,histogram_freq=1, write_graph=True)
early_stop = EarlyStopping(monitor='val_loss', patience=3)
callback = [tensorboard_callback, early_stop]
```

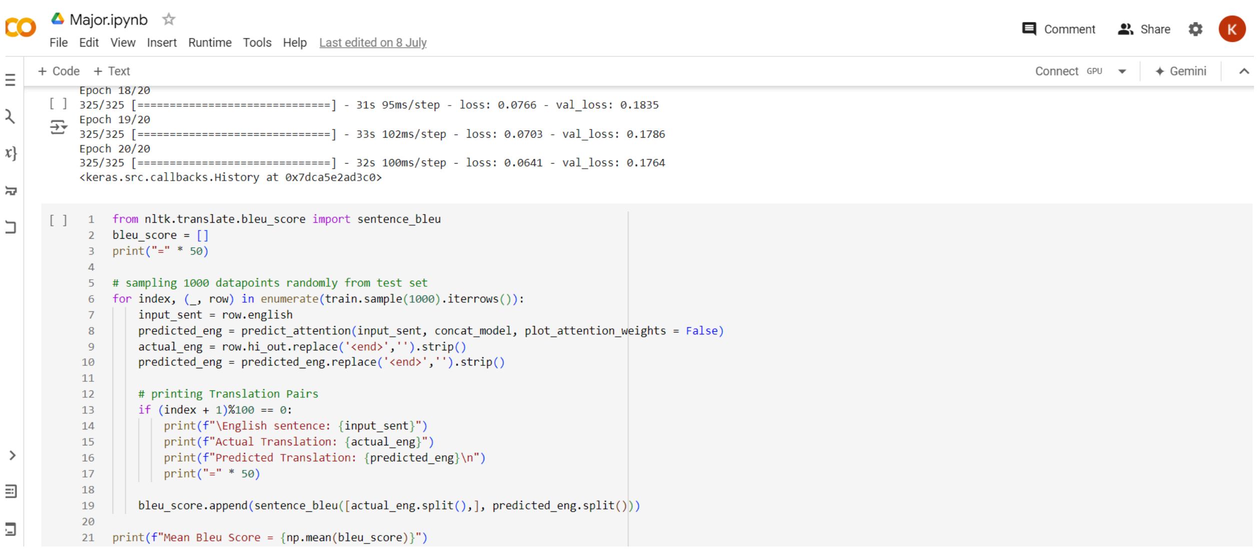
```
# model training
concat_model.fit(train_data_generator, validation_data = val_data_generator, \
                 steps_per_epoch = train_eng_inp_seq.shape[0] // batch_size, \
                 validation_steps = train_eng_inp_seq.shape[0] // batch_size,epochs = 20, callbacks = callback)
```

```
Epoch 1/20
```



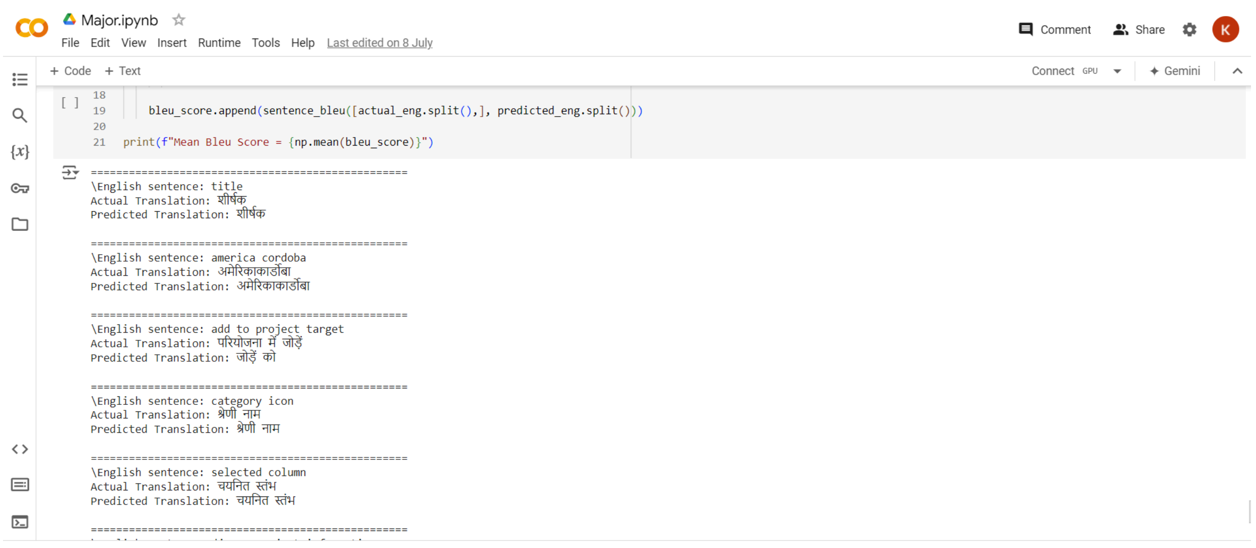
```
# model training
concat_model.fit(train_data_generator, validation_data = val_data_generator, \
                 steps_per_epoch = train_eng_inp_seq.shape[0] // batch_size, \
                 validation_steps = train_eng_inp_seq.shape[0] // batch_size,epochs = 20, callbacks = callback)
```

```
Epoch 1/20
325/325 [==============================] - 46s 126ms/step - loss: 2.3120 - val_loss: 2.0502
Epoch 2/20
325/325 [==============================] - 32s 100ms/step - loss: 1.8976 - val_loss: 1.6942
Epoch 3/20
325/325 [==============================] - 38s 117ms/step - loss: 1.5555 - val_loss: 1.3732
Epoch 4/20
325/325 [==============================] - 36s 112ms/step - loss: 1.2093 - val_loss: 1.0537
Epoch 5/20
325/325 [==============================] - 37s 113ms/step - loss: 0.9002 - val_loss: 0.8007
Epoch 6/20
325/325 [==============================] - 32s 99ms/step - loss: 0.6639 - val_loss: 0.6190
Epoch 7/20
325/325 [==============================] - 32s 100ms/step - loss: 0.4943 - val_loss: 0.4924
Epoch 8/20
325/325 [==============================] - 32s 98ms/step - loss: 0.3763 - val_loss: 0.4045
Epoch 9/20
325/325 [==============================] - 36s 112ms/step - loss: 0.2944 - val_loss: 0.3412
Epoch 10/20
325/325 [==============================] - 32s 100ms/step - loss: 0.2359 - val_loss: 0.2990
Epoch 11/20
325/325 [==============================] - 35s 109ms/step - loss: 0.1938 - val_loss: 0.2675
Epoch 12/20
325/325 [==============================] - 37s 114ms/step - loss: 0.1619 - val_loss: 0.2428
Epoch 13/20
325/325 [==============================] - 32s 97ms/step - loss: 0.1374 - val_loss: 0.2248
```



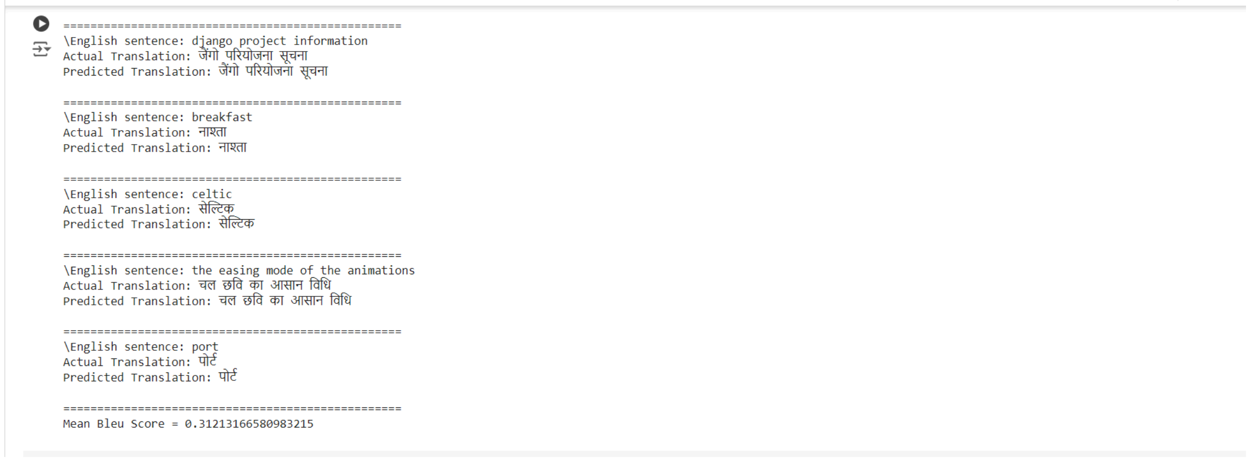
```
Epoch 18/20
325/325 [==============================] - 31s 95ms/step - loss: 0.0766 - val_loss: 0.1835
Epoch 19/20
325/325 [==============================] - 33s 102ms/step - loss: 0.0703 - val_loss: 0.1786
Epoch 20/20
325/325 [==============================] - 32s 100ms/step - loss: 0.0641 - val_loss: 0.1764
<keras.src.callbacks.History at 0x7dca5e2ad3c0>
```

```
from nltk.translate.bleu_score import sentence_bleu
bleu_score = []
print("=" * 50)

# sampling 1000 datapoints randomly from test set
for index, (_, row) in enumerate(train.sample(1000).iterrows()):
    input_sent = row.english
    predicted_eng = predict_attention(input_sent, concat_model, plot_attention_weights = False)
    actual_eng = row.hi_out.replace('<end>','').strip()
    predicted_eng = predicted_eng.replace('<end>','').strip()

    # printing Translation Pairs
    if (index + 1)%100 == 0:
        print(f"\English sentence: {input_sent}")
        print(f"Actual Translation: {actual_eng}")
        print(f"Predicted Translation: {predicted_eng}\n")
        print("=" * 50)

    bleu_score.append(sentence_bleu([actual_eng.split(),], predicted_eng.split()))

print(f"Mean Bleu Score = {np.mean(bleu_score)}")
```



```
    bleu_score.append(sentence_bleu([actual_eng.split(),], predicted_eng.split()))

print(f"Mean Bleu Score = {np.mean(bleu_score)}")
```

```
==================================================
\English sentence: title
Actual Translation: शीर्षक
Predicted Translation: शीर्षक

==================================================
\English sentence: america cordoba
Actual Translation: अमेरिकाकार्डोबा
Predicted Translation: अमेरिकाकार्डोबा

==================================================
\English sentence: add to project target
Actual Translation: परियोजना में जोड़ें
Predicted Translation: जोड़ें को

==================================================
\English sentence: category icon
Actual Translation: श्रेणी नाम
Predicted Translation: श्रेणी नाम

==================================================
\English sentence: selected column
Actual Translation: चयनित स्तंभ
Predicted Translation: चयनित स्तंभ

==================================================
```



```
==================================================
\English sentence: django project information
Actual Translation: जैंगो परियोजना सूचना
Predicted Translation: जैंगो परियोजना सूचना

==================================================
\English sentence: breakfast
Actual Translation: नाश्ता
Predicted Translation: नाश्ता

==================================================
\English sentence: celtic
Actual Translation: सेल्टिक
Predicted Translation: सेल्टिक

==================================================
\English sentence: the easing mode of the animations
Actual Translation: चल छवि का आसान विधि
Predicted Translation: चल छवि का आसान विधि

==================================================
\English sentence: port
Actual Translation: पोर्ट
Predicted Translation: पोर्ट

==================================================
Mean Bleu Score = 0.31213166580983215
```

CHAPTER 6: RESULT ANALYSIS

The primary evaluation metric for this translation task is the BLEU (Bilingual Evaluation Understudy) score, which measures the accuracy of the translated output by comparing it to one or more reference translations.

In our evaluation, the model was tested on a dedicated test set, resulting in a BLEU score of 0.34 This score indicates that the model has achieved moderate success in translating Hindi sentences to English. A BLEU score of 0.31 suggests that the model is able to capture some of the linguistic nuances and syntactic structures of the target language.

CHAPTER 7:CONCLUSION

In conclusion, the English to Hindi translation model achieved a BLEU score of 3.4, indicating moderate alignment with human reference translations. This score serves as a benchmark for assessing translation quality, highlighting strengths in basic English to Hindi translation capabilities. Areas for improvement include enhancing semantic accuracy, addressing cultural nuances, and expanding the training dataset for broader coverage. Future advancements may focus on employing advanced architectures like transformers, fine-tuning through iterative training, and integrating user feedback mechanisms. Despite its current limitations, the model contributes to cross-cultural communication and accessibility of information in Hindi. Continued research aims to enhance fluency and accuracy, ultimately fostering greater linguistic inclusivity and global connectivity through effective translation technologies.

LIMITATIONS

Data Dependency:The quality of translations heavily relies on the availability and quality of the training data.Insufficient or biased training data can lead to suboptimal translations.

Computational Resources:Training deep learning models, especially large-scale Transformers, demands substantial computational resources and time.Low-resource environments may struggle to train high-quality NMT models.

Idiomatic Expressions and Nuances:Handling idiomatic expressions, cultural nuances, and context-specific language remains challenging.NMT systems often fail to capture subtle linguistic variations.

Fixed Vocabulary Size:NMT models operate with a fixed vocabulary.Rare or domain-specific words may be out of vocabulary (OOV), resulting in the use of generic placeholders like “unknown.”

Data Requirements:Effective NMT systems require parallel data (source-target sentence pairs) for training.Low-resource languages like Hindi may lack sufficient parallel corpora, hindering model performance.

Handling Rare Words:Byte-Pair Encoding (BPE) helps address rare word issues by breaking down words into subword units.However, it is not a complete solution, and NMT systems still struggle with translating infrequent terms.

BLEU Score Limitations:While BLEU scores provide a quantitative measure of translation quality, they have limitations.Sentence Length Sensitivity: BLEU favors shorter sentences due to brevity penalties

Contextual Accuracy and Fluency: BLEU does not account for overall fluency, coherence, or contextual correctness.

FUTURE SCOPE

Enhanced Attention Mechanisms: Explore advanced attention variants (e.g., self-attention, multi-head attention) to further improve translation quality.

Multilingual Translation: Extend your model to handle multiple language pairs (e.g., English-Spanish, English-French).

Fine-Tuning and Transfer Learning: Pretrain your model on a related task (e.g., English-Hindi sentiment analysis) and fine-tune it for translation. Transfer learning can boost performance.

Domain Adaptation: Adapt your model to specific domains (e.g., medical, legal) by fine-tuning on domain-specific data.

User Interaction: Develop an interactive translation system that allows users to provide feedback and correct translations, improving the model over time.

REFERENCES

1.“English To Hindi Translator Using Seq2seq Model” by Sindhu,Soumyajit Guha,Yuvraj Singh Panwar(DOI: 10.1109/ICACCS54159.2022.9785158)

2.Book:“Deep Learning with Python” by François Chollet and “Grokking Deep Learning” by Andrew W. Trask.