Experiment No 4

Jhanvi Parekh

60009210033

D11

Aim: Implement Language translator using Encoder Decoder model.

Theory:

Introduction

The encoder-decoder model is a way of using recurrent neural networks for sequence-tosequence prediction problems.

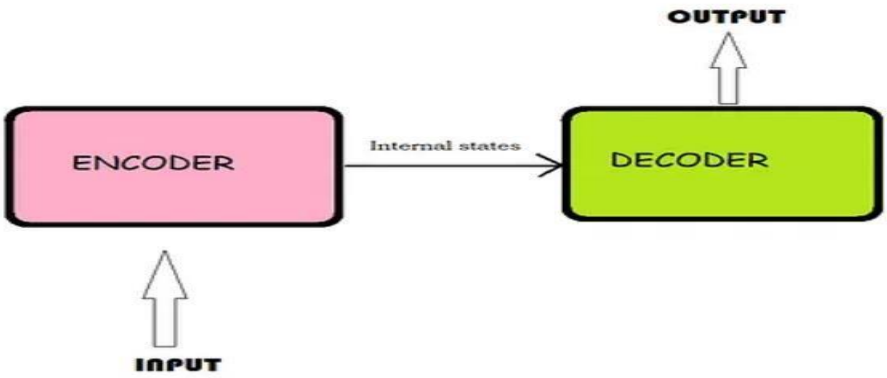
It was initially developed for machine translation problems, although it has proven successful at related sequence-to-sequence prediction problems such as text summarization and question answering.

The approach involves two recurrent neural networks, one to encode the input sequence, called the encoder, and a second to decode the encoded input sequence into the target sequence called the decoder.

Following are some of the application of sequence to sequence models-

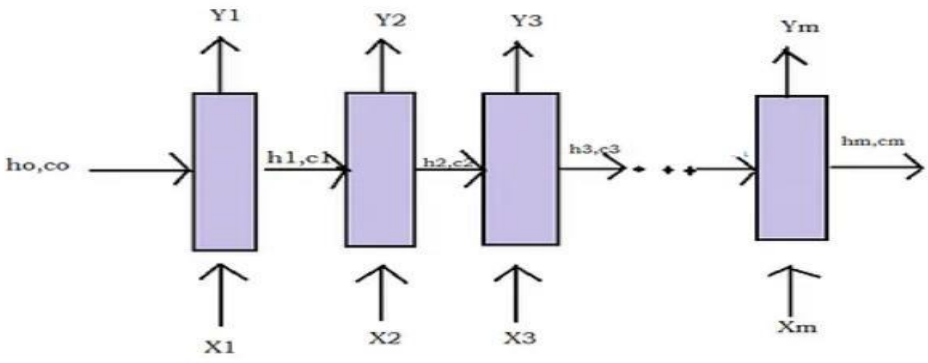
- Chatbots
- Machine Translation
- Text summary
- Image captioning

The architecture of Encoder-Decoder : The overall structure of sequence to sequence model(encoder-decoder) which is commonly used is as shown below



It consists of 3 parts: **encoder, intermediate vector and decoder.**

- **Encoder-**It accepts a single element of the input sequence at each time step, process it, collects information for that element and propagates it forward.
- **Intermediate vector**- This is the final internal state produced from the encoder part of the model. It contains information about the entire input sequence to help the decoder make accurate predictions.
- **Decoder-** given the entire sentence, it predicts an output at each time step. ● **Understanding the Encoder part of the model** ● The encoder is basically an LSTM/GRU cell.
- An encoder takes the input sequence and encapsulates the information as the internal state vectors.
- Outputs of the encoder are rejected and only internal states are used.
- Let's understand how encoder part of the model works



LSTM for encoder

LSTM takes only one element at a time, so if the input sequence is of length m, then LSTM takes m time steps to read the entire sequence.

- Xt is the input at time step t.
- ht and ct are internal states at time step t of the LSTM and for GRU there is only one internal state ht.
- Yt is the output at time step t.

Let's consider an example of English to Hindi translation

Inputs of Encoder Xt-

Consider the English sentence- India is beautiful country. This sequence can be thought of as a sentence containing 4 words (India, is, beautiful, country). So here

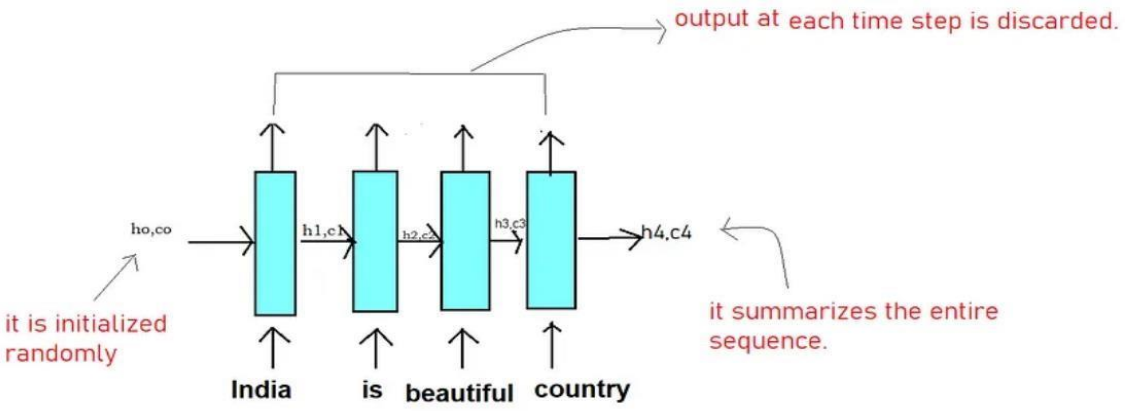
X1 =’India’

X2=’is’ X3=

‘beautiful’

X4=’country’.

Therefore LSTM will read this sequence word by word in 4-time step as follows



Here each Xt (each word) is represented as a vector using the word embedding, which converts each word into a vector of fixed length. **Now coming to internal states (ht,ct) -**

- It learns what the LSTM has read until time step t. For e.g when t=2, it remembers that LSTM has read ‘*India is ‘.*
- The initial states ho, co(both are vectors) is initialized randomly or with zeroes.
- Remember the dimension of ho, co is same as the number of units in LSTM cell.
- The final state h4,c4 contains the crux of the entire input sequence I*ndia is beautiful country.*

The output of encoder Yt-

Yt at each time steps is the predictions of the LSTM at each time step. In machine translation problems, we generate the outputs when we have read the entire input sequence. So Yt at each time step in the encoder is of no use so we discard it.

The encoder will read the English sentence word by word and store the final internal states (known as an intermediate vector) of the LSTM generated after the last time step and since the output will be generated once the entire sequence is read, therefore outputs (Yt) of the Encoder at each time step are discarded.

Understanding the Decoder part of the model in the Training Phase-

The working of the decoder is different during the training and testing phase unlike the encoder part of the model which works in the same fashion in training and test phase.

Let's understand the working of the decoder part during training phase-

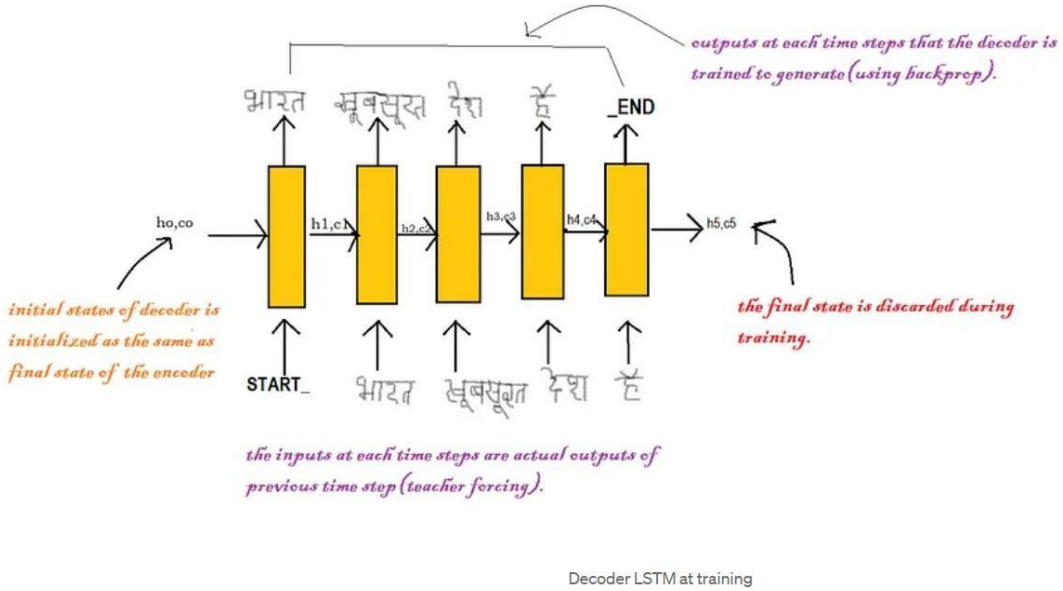
Taking the running example of translating ***India is beautiful country*** to its Hindi counterpart, just like encoder, the decoder also generates the output sentence word by word.

So we want to generate the output — **भारत खूबसूरत देश है**

For the decoder to recognize the starting and end of the sequence, we will add START_ at the beginning of the output sequence and _END at the end of the output sequence.

So our Output sentence will be ***START_*** **भारत खूबसूरत देश है** ***_END***

Let's understand the working visually

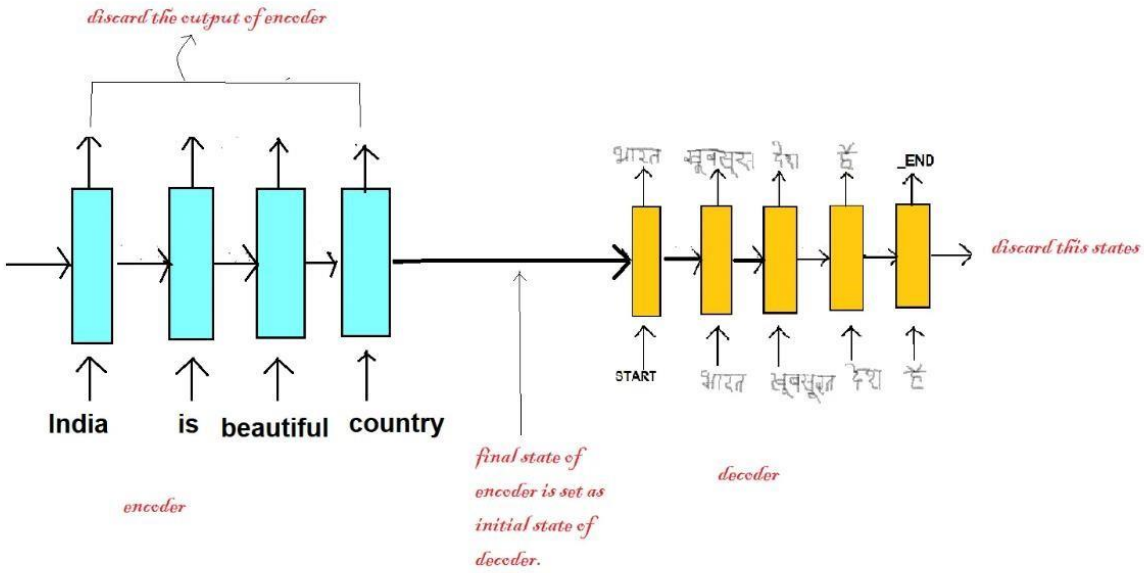


Decoder LSTM at training

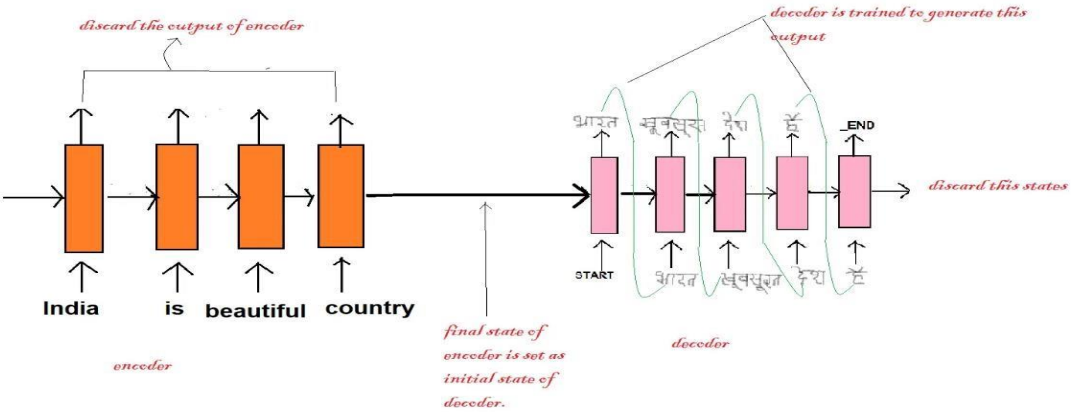
- The initial states (ho, co) of the decoder is set to the final states of the encoder. It can be thought of as that the decoder is trained to generate the output based on the information gathered by the encoder.

- Firstly, we input the START_ so that the decoder starts generating the next word. And after the last word in the Hindi sentence, we make the decoder learn to predict the _END.
- Here we use the teacher forcing technique where the input at each time step is actual output and not the predicted output from the last time step.
- At last, the loss is calculated on the predicted outputs from each time step and the errors are backpropagated through time to update the parameters of the model.
- The final states of the decoder are discarded as we got the output hence it is of no use.

Summarizing the encoder-decoder visually



Understanding the Decoder part of the model in Test Phase



Process of the Decoder in the test period-

- The initial states of the decoder are set to the final states of the encoder.
- LSTM in the decoder process single word at every time step.
- Input to the decoder always starts with the START_.
- The internal states generated after every time step is fed as the initial states of the next time step. for e.g At t=1, the internal states produced after inputting START_ is fed as the initial states at t=2.
- The output produced at each time step is fed as input in the next time step.
- We get to know about the end of the sequence when the decoder predicts the END_.

Lab Experiment to be performed in the session :

Exercise-1: Create a base encoder-decoder machine translation model with LSTM

Step 1:

Use a dataset of pairs of English sentences and their Marathi translation, which you can download from manythings.org/anki. There are several other translations available, you can choose whichever you like. The file to download is called mar-eng.zip.

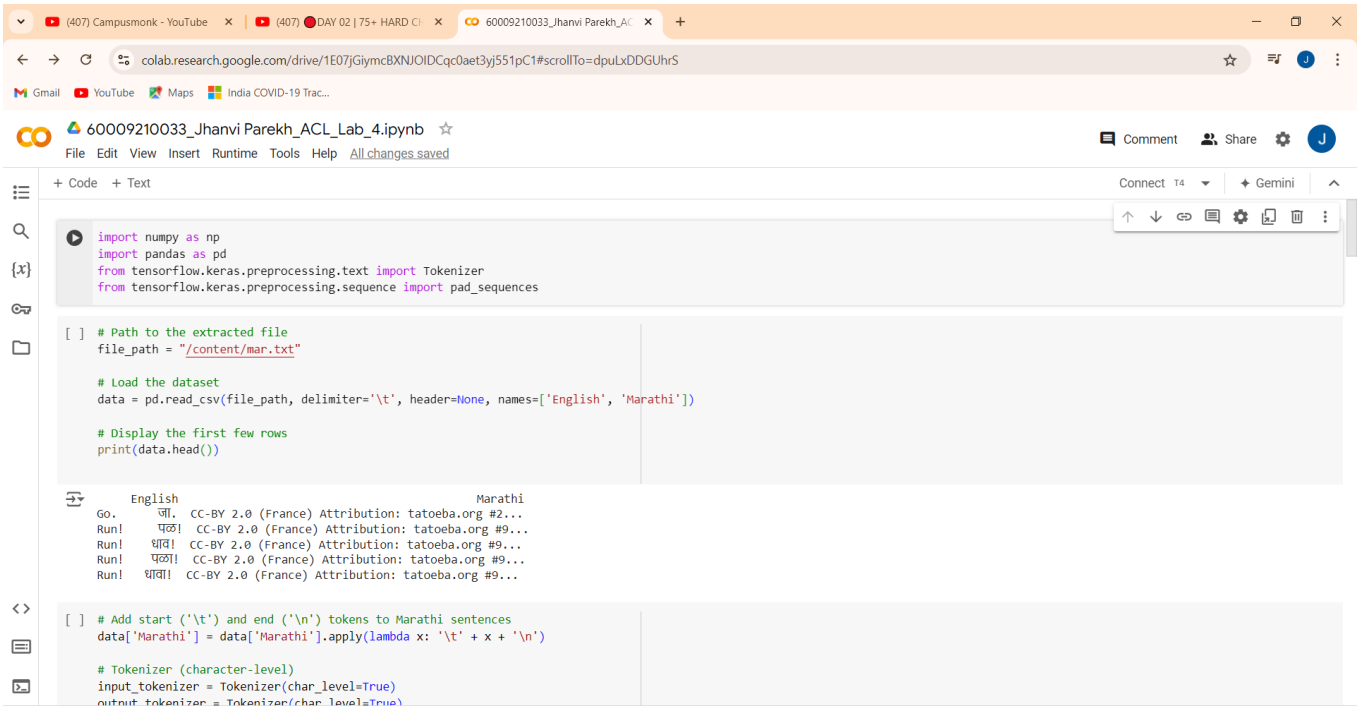
Step 2:

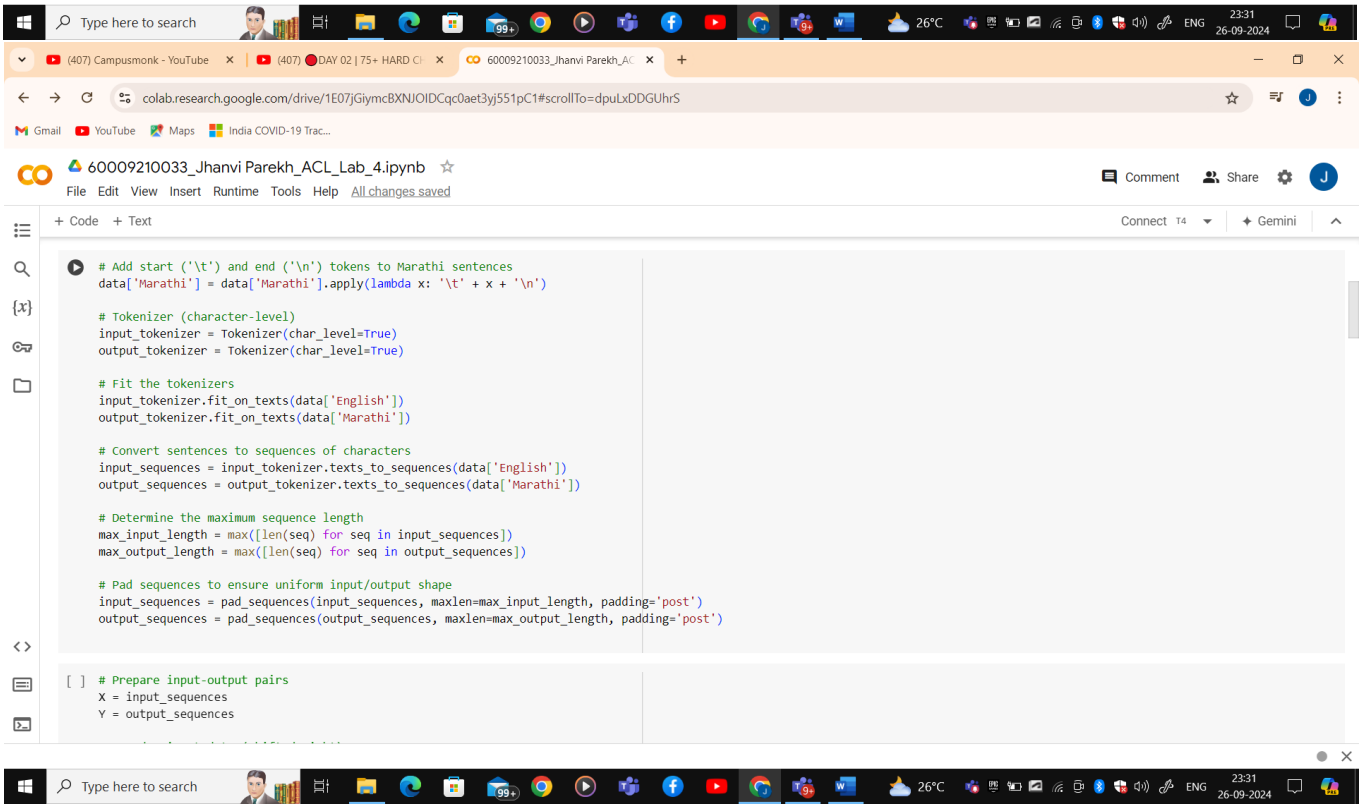
Implement a *character-level* sequence-to-sequence model, processing the input character-bycharacter and generating the output character-by-character.

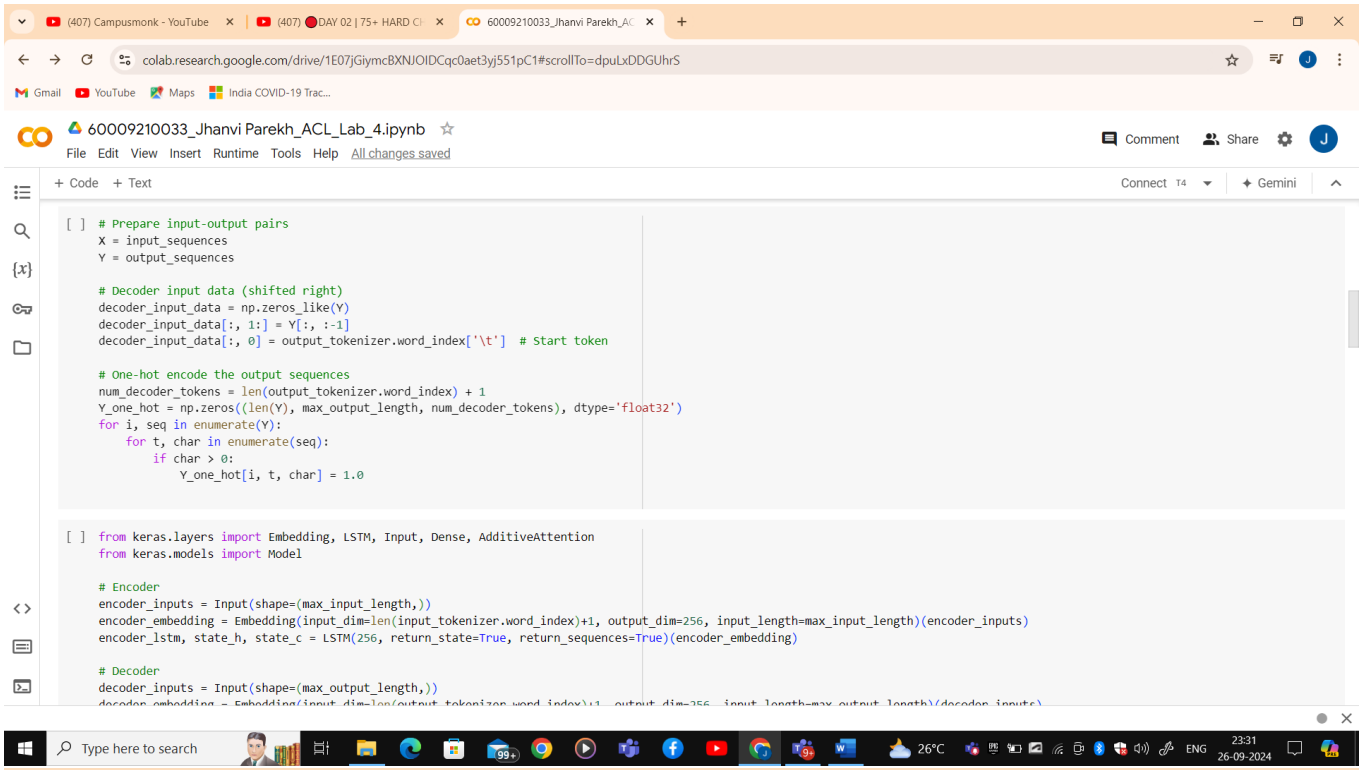
OR

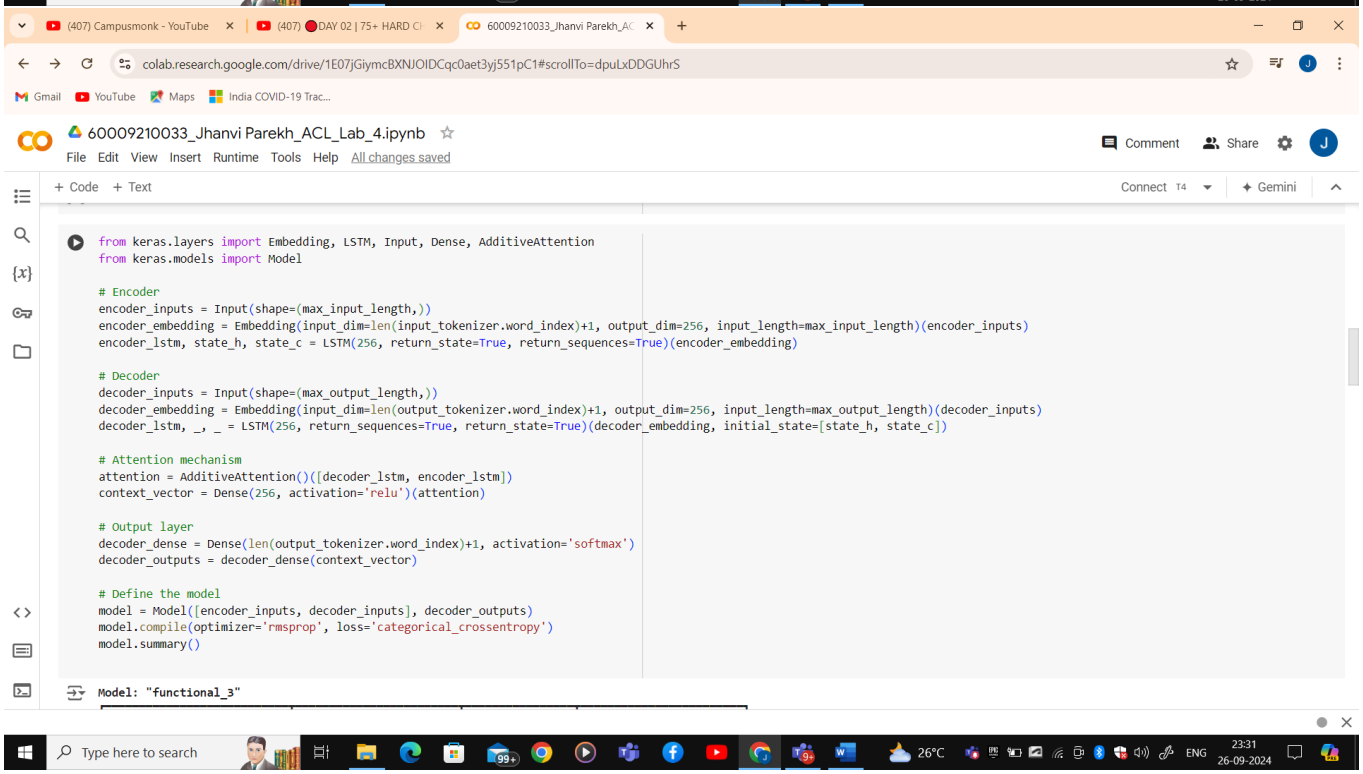
Implement a word-level model, which tends to be more common for machine translation.

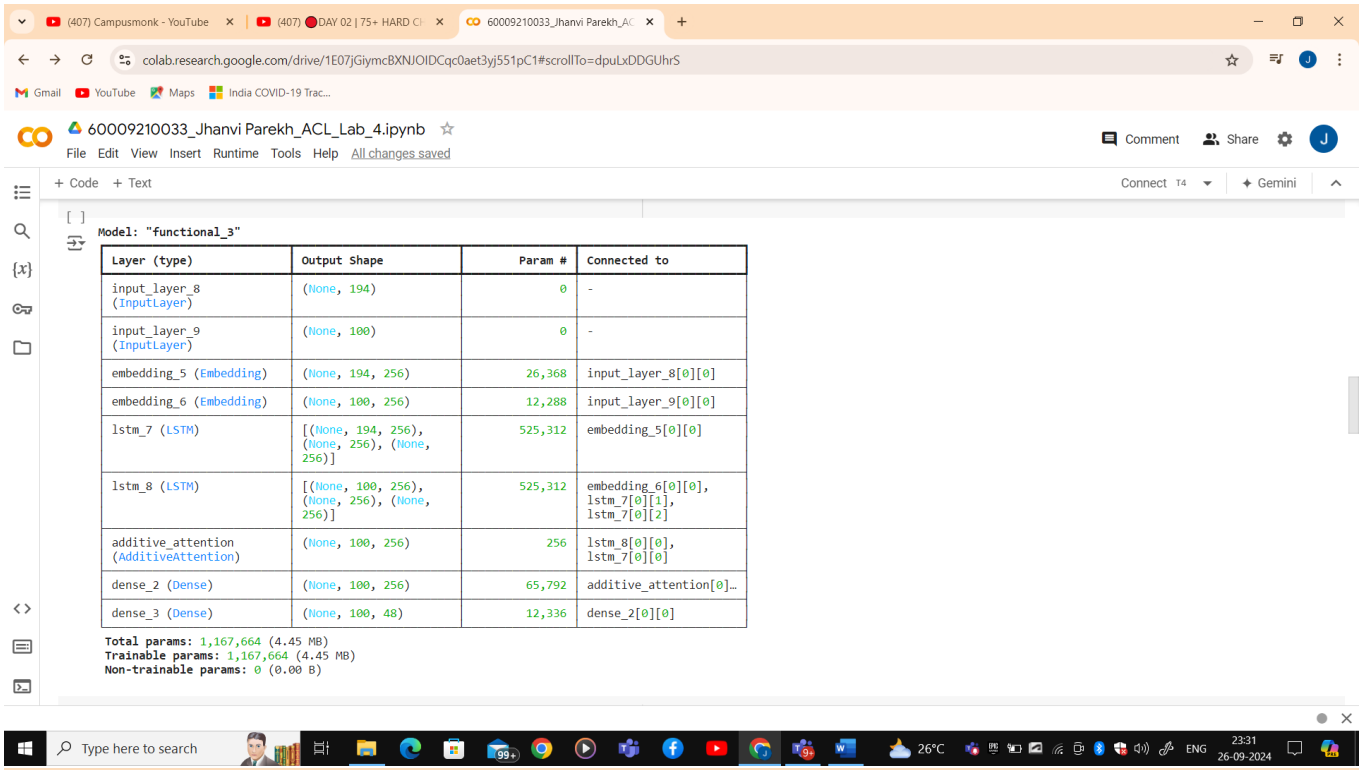
Colab link:
https://colab.research.google.com/drive/1E07jGivmcBXNJOIDCqc0aet3yj551pC1?usp=sharing


(407) Campusmonk - YouTube
(407) DAY 02 | 75+ HARD Ch
60009210033_Jhanvi Parekh_AC
colab.research.google.com/drive/1E07jGiymcBXNJOIDCqc0aet3yj551pC1#scrollTo=dpuLxDDGUhrS
Gmail
YouTube
Maps
India COVID-19 Trac...
60009210033_Jhanvi Parekh_ACL_Lab_4.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect T4
Gemini
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
[ ] # Path to the extracted file
file_path = "/content/mar.txt"
# Load the dataset
data = pd.read_csv(file_path, delimiter='\t', header=None, names=['English', 'Marathi'])
# Display the first few rows
print(data.head())
English Marathi
Go. जा. CC-BY 2.0 (France) Attribution: tatoeba.org #2...
Run! पळ! CC-BY 2.0 (France) Attribution: tatoeba.org #9...
Run! धाव! CC-BY 2.0 (France) Attribution: tatoeba.org #9...
Run! पळा! CC-BY 2.0 (France) Attribution: tatoeba.org #9...
Run! धावा! CC-BY 2.0 (France) Attribution: tatoeba.org #9...
[ ] # Add start ('\t') and end ('\n') tokens to Marathi sentences
data['Marathi'] = data['Marathi'].apply(lambda x: '\t' + x + '\n')
# Tokenizer (character-level)
input_tokenizer = Tokenizer(char_level=True)
output_tokenizer = Tokenizer(char_level=True)


(407) Campusmonk - YouTube
(407) DAY 02 | 75+ HARD Ch
60009210033_Jhanvi Parekh_AC
colab.research.google.com/drive/1E07jGiymcBXNJOIDCqc0aet3yj551pC1#scrollTo=dpuLxDDGUhrS
Gmail
YouTube
Maps
India COVID-19 Trac...
60009210033_Jhanvi Parekh_ACL_Lab_4.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect T4
Gemini
Add start ('\t') and end ('\n') tokens to Marathi sentences
data['Marathi'] = data['Marathi'].apply(lambda x: '\t' + x + '\n')
# Tokenizer (character-level)
input_tokenizer = Tokenizer(char_level=True)
output_tokenizer = Tokenizer(char_level=True)
# Fit the tokenizers
input_tokenizer.fit_on_texts(data['English'])
output_tokenizer.fit_on_texts(data['Marathi'])
# Convert sentences to sequences of characters
input_sequences = input_tokenizer.texts_to_sequences(data['English'])
output_sequences = output_tokenizer.texts_to_sequences(data['Marathi'])
# Determine the maximum sequence length
max_input_length = max([len(seq) for seq in input_sequences])
max_output_length = max([len(seq) for seq in output_sequences])
# Pad sequences to ensure uniform input/output shape
input_sequences = pad_sequences(input_sequences, maxlen=max_input_length, padding='post')
output_sequences = pad_sequences(output_sequences, maxlen=max_output_length, padding='post')
[ ] # Prepare input-output pairs
X = input_sequences
Y = output_sequences


(407) Campusmonk - YouTube
(407) DAY 02 | 75+ HARD Cl
60009210033_Jhanvi Parekh_AC
colab.research.google.com/drive/1E07jGiymcBXNJOIDCqc0aet3yj551pC1#scrollTo=dpuLxDDGUhrS
Gmail
YouTube
Maps
India COVID-19 Trac...
60009210033_Jhanvi Parekh_ACL_Lab_4.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect T4
Gemini
[ ] # Prepare input-output pairs
X = input_sequences
Y = output_sequences
# Decoder input data (shifted right)
decoder_input_data = np.zeros_like(Y)
decoder_input_data[:, 1:] = Y[:, :-1]
decoder_input_data[:, 0] = output_tokenizer.word_index['\t'] # Start token
# One-hot encode the output sequences
num_decoder_tokens = len(output_tokenizer.word_index) + 1
Y_one_hot = np.zeros((len(Y), max_output_length, num_decoder_tokens), dtype='float32')
for i, seq in enumerate(Y):
for t, char in enumerate(seq):
if char > 0:
Y_one_hot[i, t, char] = 1.0
[ ] from keras.layers import Embedding, LSTM, Input, Dense, AdditiveAttention
from keras.models import Model
# Encoder
encoder_inputs = Input(shape=(max_input_length,))
encoder_embedding = Embedding(input_dim=len(input_tokenizer.word_index)+1, output_dim=256, input_length=max_input_length)(encoder_inputs)
encoder_lstm, state_h, state_c = LSTM(256, return_state=True, return_sequences=True)(encoder_embedding)
# Decoder
decoder_inputs = Input(shape=(max_output_length,))
Type here to search
26°C
ENG
23:31
26-09-2024


(407) Campusmonk - YouTube
(407) DAY 02 | 75+ HARD Cl
60009210033_Jhanvi Parekh_AC
colab.research.google.com/drive/1E07jGiymcBXNJOIDCqc0aet3yj551pC1#scrollTo=dpuLxDDGUhrS
Gmail
YouTube
Maps
India COVID-19 Trac...
60009210033_Jhanvi Parekh_ACL_Lab_4.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect T4
Gemini
from keras.layers import Embedding, LSTM, Input, Dense, AdditiveAttention
from keras.models import Model
# Encoder
encoder_inputs = Input(shape=(max_input_length,))
encoder_embedding = Embedding(input_dim=len(input_tokenizer.word_index)+1, output_dim=256, input_length=max_input_length)(encoder_inputs)
encoder_lstm, state_h, state_c = LSTM(256, return_state=True, return_sequences=True)(encoder_embedding)
# Decoder
decoder_inputs = Input(shape=(max_output_length,))
decoder_embedding = Embedding(input_dim=len(output_tokenizer.word_index)+1, output_dim=256, input_length=max_output_length)(decoder_inputs)
decoder_lstm, _, _ = LSTM(256, return_sequences=True, return_state=True)(decoder_embedding, initial_state=[state_h, state_c])
# Attention mechanism
attention = AdditiveAttention()([decoder_lstm, encoder_lstm])
context_vector = Dense(256, activation='relu')(attention)
# Output layer
decoder_dense = Dense(len(output_tokenizer.word_index)+1, activation='softmax')
decoder_outputs = decoder_dense(context_vector)
# Define the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
model.summary()
Model: "functional_3"
Type here to search
26°C
ENG
23:31
26-09-2024



60009210033_Jhanvi Parekh_ACL_Lab_4.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

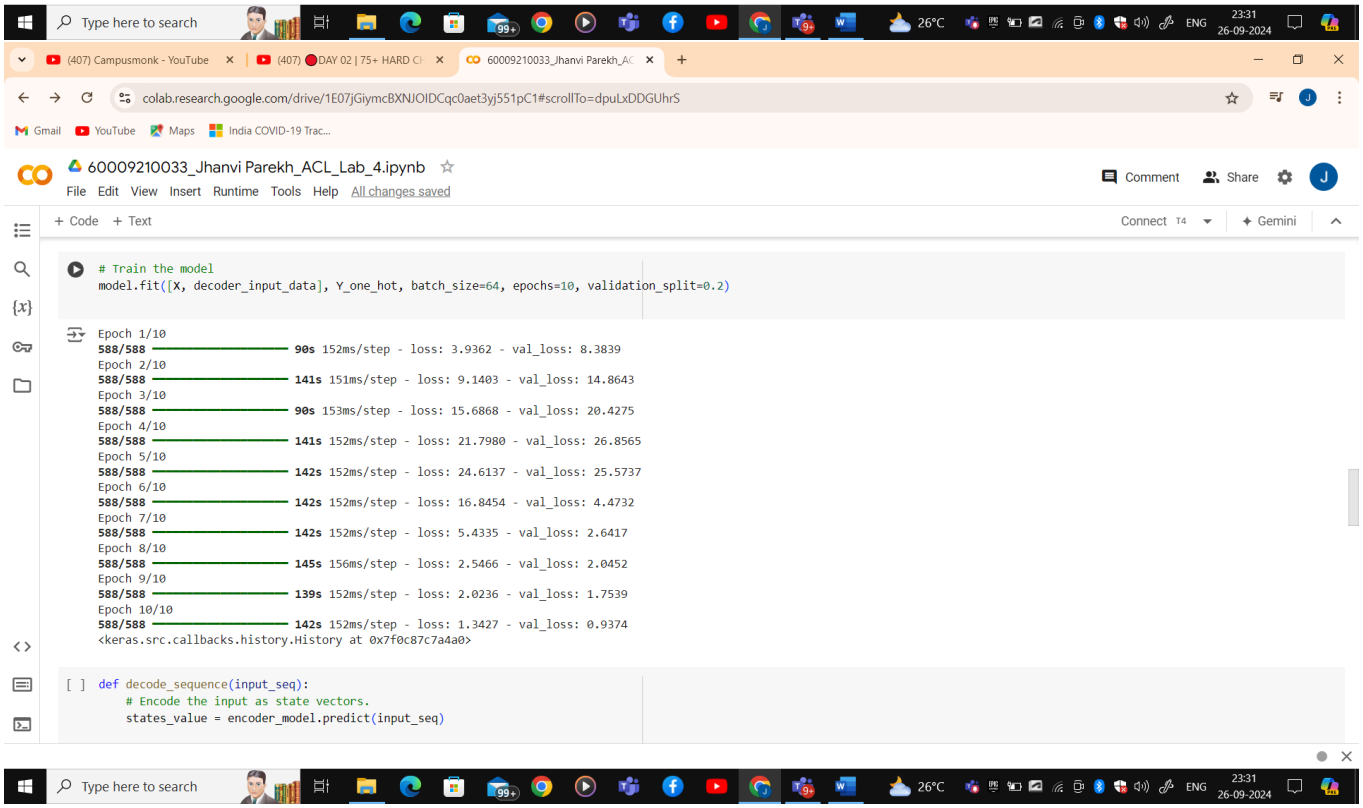
Connect T4 Gemini

[ ]

Model: "functional_3"

Layer (type)	Output Shape	Param #	Connected to
input_layer_8 (InputLayer)	(None, 194)	0	-
input_layer_9 (InputLayer)	(None, 100)	0	-
embedding_5 (Embedding)	(None, 194, 256)	26,368	input_layer_8[0][0]
embedding_6 (Embedding)	(None, 100, 256)	12,288	input_layer_9[0][0]
lstm_7 (LSTM)	[(None, 194, 256), (None, 256), (None, 256)]	525,312	embedding_5[0][0]
lstm_8 (LSTM)	[(None, 100, 256), (None, 256), (None, 256)]	525,312	embedding_6[0][0], lstm_7[0][1], lstm_7[0][2]
additive_attention (AdditiveAttention)	(None, 100, 256)	256	lstm_8[0][0], lstm_7[0][0]
dense_2 (Dense)	(None, 100, 256)	65,792	additive_attention[0]…
dense_3 (Dense)	(None, 100, 48)	12,336	dense_2[0][0]

Total params: 1,167,664 (4.45 MB)
Trainable params: 1,167,664 (4.45 MB)
Non-trainable params: 0 (0.00 B)



60009210033_Jhanvi Parekh_ACL_Lab_4.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

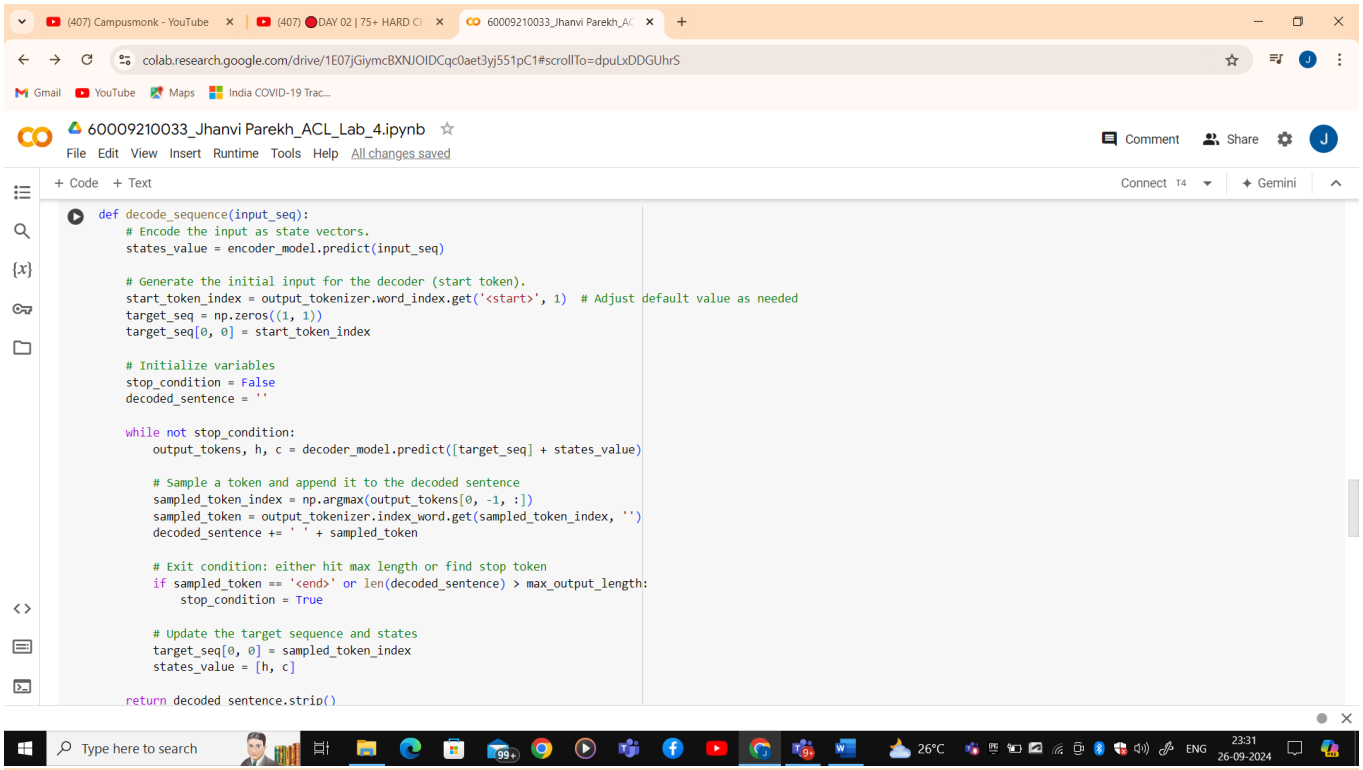
+ Code + Text

Connect T4 Gemini

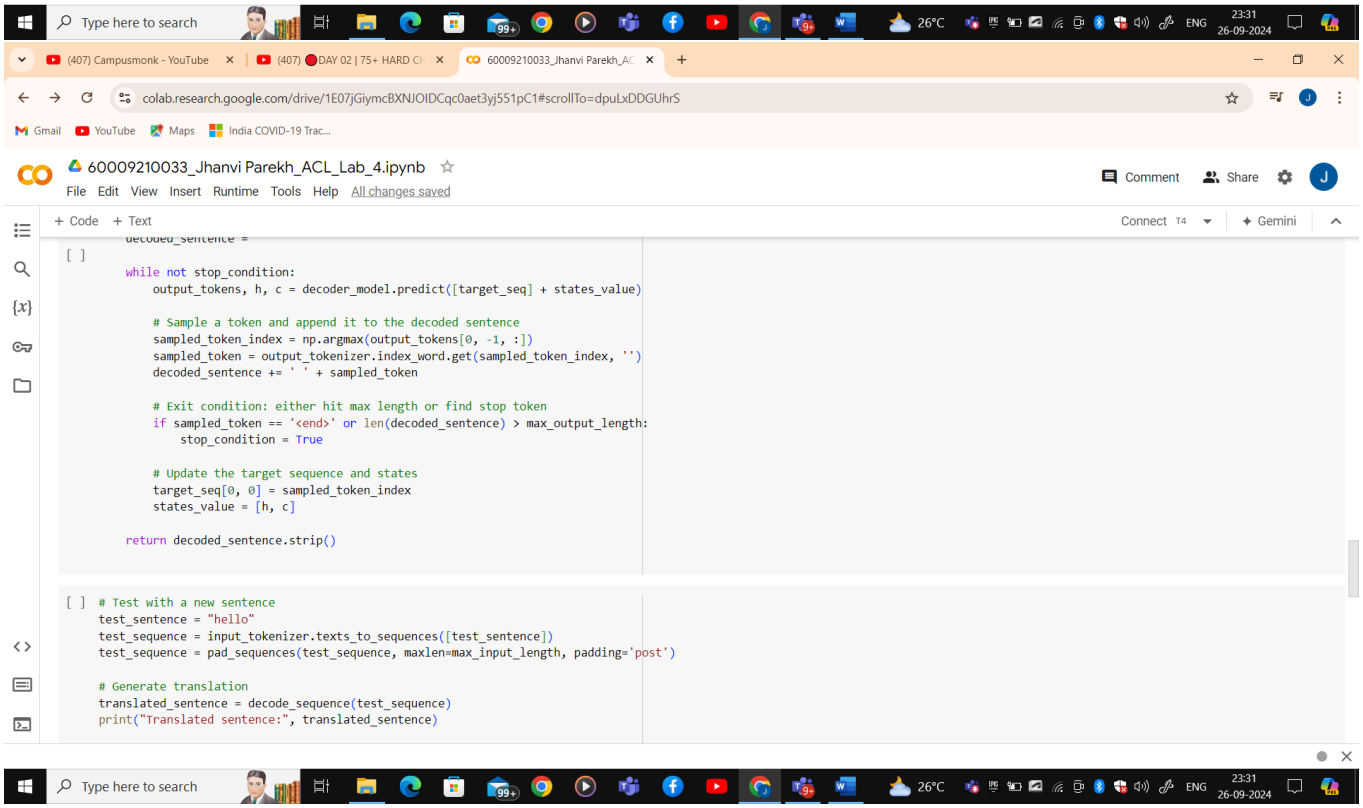
```
# Train the model
model.fit([X, decoder_input_data], Y_one_hot, batch_size=64, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
588/588 ━━━━━━━━━━━━━━━━━━━━ 90s 152ms/step - loss: 3.9362 - val_loss: 8.3839
Epoch 2/10
588/588 ━━━━━━━━━━━━━━━━━━━━ 141s 151ms/step - loss: 9.1403 - val_loss: 14.8643
Epoch 3/10
588/588 ━━━━━━━━━━━━━━━━━━━━ 90s 153ms/step - loss: 15.6868 - val_loss: 20.4275
Epoch 4/10
588/588 ━━━━━━━━━━━━━━━━━━━━ 141s 152ms/step - loss: 21.7980 - val_loss: 26.8565
Epoch 5/10
588/588 ━━━━━━━━━━━━━━━━━━━━ 142s 152ms/step - loss: 24.6137 - val_loss: 25.5737
Epoch 6/10
588/588 ━━━━━━━━━━━━━━━━━━━━ 142s 152ms/step - loss: 16.8454 - val_loss: 4.4732
Epoch 7/10
588/588 ━━━━━━━━━━━━━━━━━━━━ 142s 152ms/step - loss: 5.4335 - val_loss: 2.6417
Epoch 8/10
588/588 ━━━━━━━━━━━━━━━━━━━━ 145s 156ms/step - loss: 2.5466 - val_loss: 2.0452
Epoch 9/10
588/588 ━━━━━━━━━━━━━━━━━━━━ 139s 152ms/step - loss: 2.0236 - val_loss: 1.7539
Epoch 10/10
588/588 ━━━━━━━━━━━━━━━━━━━━ 142s 152ms/step - loss: 1.3427 - val_loss: 0.9374
<keras.src.callbacks.history.History at 0x7f0c87c7a4a0>
```

```
[ ] def decode_sequence(input_seq):
        # Encode the input as state vectors.
        states_value = encoder_model.predict(input_seq)
```

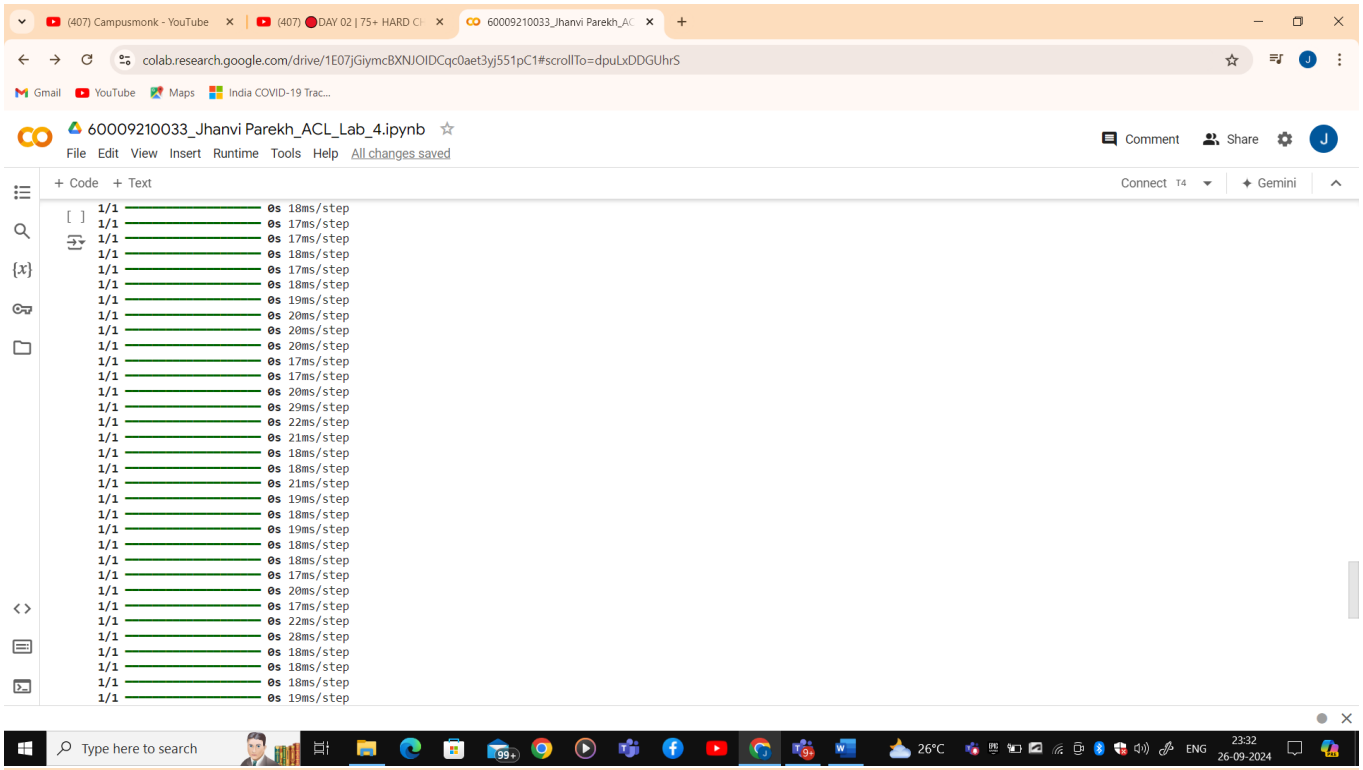


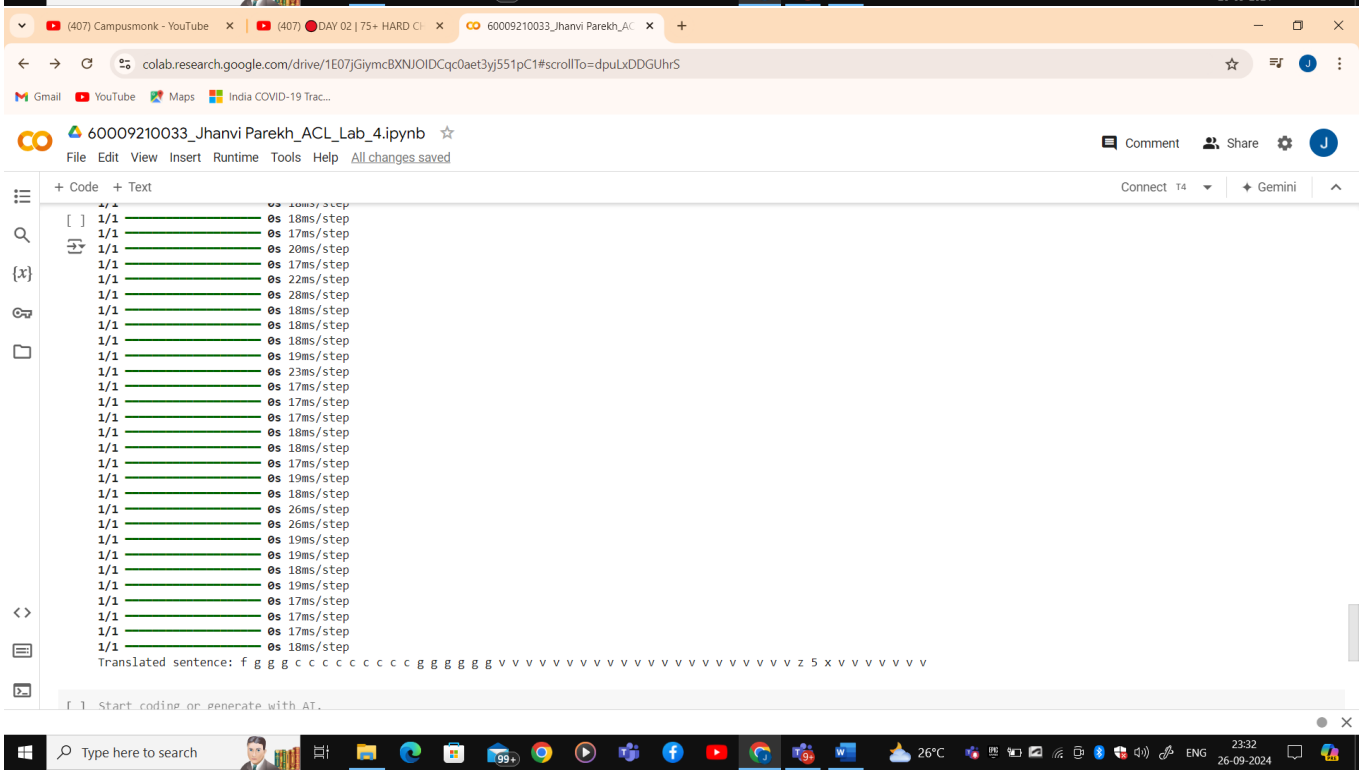
```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    states_value = encoder_model.predict(input_seq)

    # Generate the initial input for the decoder (start token).
    start_token_index = output_tokenizer.word_index.get('<start>', 1)  # Adjust default value as needed
    target_seq = np.zeros((1, 1))
    target_seq[0, 0] = start_token_index

    # Initialize variables
    stop_condition = False
    decoded_sentence = ''

    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)

        # Sample a token and append it to the decoded sentence
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_token = output_tokenizer.index_word.get(sampled_token_index, '')
        decoded_sentence += ' ' + sampled_token

        # Exit condition: either hit max length or find stop token
        if sampled_token == '<end>' or len(decoded_sentence) > max_output_length:
            stop_condition = True

        # Update the target sequence and states
        target_seq[0, 0] = sampled_token_index
        states_value = [h, c]

    return decoded_sentence.strip()
```



```
[ ]
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)

        # Sample a token and append it to the decoded sentence
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_token = output_tokenizer.index_word.get(sampled_token_index, '')
        decoded_sentence += ' ' + sampled_token

        # Exit condition: either hit max length or find stop token
        if sampled_token == '<end>' or len(decoded_sentence) > max_output_length:
            stop_condition = True

        # Update the target sequence and states
        target_seq[0, 0] = sampled_token_index
        states_value = [h, c]

    return decoded_sentence.strip()
```

```
[ ] # Test with a new sentence
    test_sentence = "hello"
    test_sequence = input_tokenizer.texts_to_sequences([test_sentence])
    test_sequence = pad_sequences(test_sequence, maxlen=max_input_length, padding='post')

    # Generate translation
    translated_sentence = decode_sequence(test_sequence)
    print("Translated sentence:", translated_sentence)
```


(407) Campusmonk - YouTube
(407) DAY 02 | 75+ HARD Cl
60009210033_Jhanvi Parekh_AC
colab.research.google.com/drive/1E07jGiymcBXNJOIDCqc0aet3yj551pC1#scrollTo=dpuLxDDGUhrS
Gmail
YouTube
Maps
India COVID-19 Trac...
60009210033_Jhanvi Parekh_ACL_Lab_4.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect T4
Gemini
[ ]
1/1 0s 18ms/step
1/1 0s 17ms/step
1/1 0s 17ms/step
1/1 0s 18ms/step
1/1 0s 17ms/step
1/1 0s 18ms/step
1/1 0s 19ms/step
1/1 0s 20ms/step
1/1 0s 20ms/step
1/1 0s 20ms/step
1/1 0s 17ms/step
1/1 0s 17ms/step
1/1 0s 20ms/step
1/1 0s 29ms/step
1/1 0s 22ms/step
1/1 0s 21ms/step
1/1 0s 18ms/step
1/1 0s 18ms/step
1/1 0s 21ms/step
1/1 0s 19ms/step
1/1 0s 18ms/step
1/1 0s 19ms/step
1/1 0s 18ms/step
1/1 0s 18ms/step
1/1 0s 17ms/step
1/1 0s 20ms/step
1/1 0s 17ms/step
1/1 0s 22ms/step
1/1 0s 28ms/step
1/1 0s 18ms/step
1/1 0s 18ms/step
1/1 0s 18ms/step
Type here to search
26°C
ENG
23:32
26-09-2024


(407) Campusmonk - YouTube
(407) DAY 02 | 75+ HARD Cl
60009210033_Jhanvi Parekh_AC
colab.research.google.com/drive/1E07jGiymcBXNJOIDCqc0aet3yj551pC1#scrollTo=dpuLxDDGUhrS
Gmail
YouTube
Maps
India COVID-19 Trac...
60009210033_Jhanvi Parekh_ACL_Lab_4.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect T4
Gemini
[ ]
1/1 0s 18ms/step
1/1 0s 17ms/step
1/1 0s 20ms/step
1/1 0s 17ms/step
1/1 0s 22ms/step
1/1 0s 28ms/step
1/1 0s 18ms/step
1/1 0s 18ms/step
1/1 0s 18ms/step
1/1 0s 19ms/step
1/1 0s 23ms/step
1/1 0s 17ms/step
1/1 0s 17ms/step
1/1 0s 17ms/step
1/1 0s 18ms/step
1/1 0s 18ms/step
1/1 0s 17ms/step
1/1 0s 19ms/step
1/1 0s 18ms/step
1/1 0s 26ms/step
1/1 0s 26ms/step
1/1 0s 19ms/step
1/1 0s 19ms/step
1/1 0s 18ms/step
1/1 0s 19ms/step
1/1 0s 17ms/step
1/1 0s 17ms/step
1/1 0s 17ms/step
1/1 0s 18ms/step
Translated sentence: f g g g c c c c c c c c c g g g g g g g v v v v v v v v v v v v v v v v v v v v v v v v v v z 5 x v v v v v v v v
[ ] Start coding or generate with AI.
Type here to search
26°C
ENG
23:32
26-09-2024